UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

ENCODING THE SATISFIABILITY OF MODAL AND DESCRIPTION LOGICS INTO SAT: THE CASE STUDY OF K(M)/ALC

Roberto Sebastiani and Michele Vescovi

May 2006

Technical Report # DIT-06-033

Encoding the satisfiability of modal and description logics into SAT: the case study of K(m)/$\mathcal{ALC}$

Roberto Sebastiani and Michele Vescovi

DIT, Università di Trento, Via Sommarive 14, I-38050, Povo, Trento, Italy.
roberto.sebastiani@dit.unitn.it, michele.vescovi@studenti.unitn.it

Abstract. In the last two decades, modal and description logics have been applied to numerous areas of computer science, including artificial intelligence, formal verification, database theory, and distributed computing. For this reason, the problem of automated reasoning in modal and description logics has been throughly investigated.
In particular, many approaches have been proposed for efficiently handling the satisfiability of the core normal modal logic K_m, and of its notational variant, the description logic $\mathcal{ALC}$. Although simple in structure, $K_m/\mathcal{ALC}$ is computationally very hard to reason on, its satisfiability being PSPACE-complete.
In this paper we explore the idea of encoding $K_m/\mathcal{ALC}$-satisfiability into SAT, so that to be handled by state-of-the-art SAT tools. We propose an efficient encoding, and we test it on an extensive set of benchmarks, comparing the approach with the main state-of-the-art tools available.
Although the encoding is necessarily worst-case exponential, from our experiments we notice that, in practice, this approach can handle most or all the problems which are at the reach of the other approaches, with performances which are comparable with, or even better than, those of the current state-of-the-art tools.

1 Introduction

In the last two decades, modal and description logics have been applied to numerous areas of computer science, including artificial intelligence, formal verification, database theory, and distributed computing. For this reason, the problem of automated reasoning in modal and description logics has been throughly investigated (see, e.g., [5, 17, 10, 18]). Many approaches have been proposed for efficiently handling the satisfiability of modal and description logics, in particular of the core normal modal logic K_m and of its notational variant, the description logic $\mathcal{ALC}$ (see, e.g., [5, 18, 7, 8, 12, 15, 14, 3, 19, 20]). Notice that, although simple in structure, $K_m/\mathcal{ALC}$ is computationally very hard to reason on, as its satisfiability is PSPACE-complete [17, 10].

In this paper we explore the idea of encoding $K_m/\mathcal{ALC}$-satisfiability into SAT, so that to be handled by state-of-the-art SAT tools. We propose an efficient encoding, with four simple variations. We test (the four variations of) it on an extensive set of benchmarks, comparing the results with those of the main state-of-the-art tools for K_m-satisfiability available.

Although the encoding is necessarily worst-case exponential (unless PSPACE=NP), from our experiments we notice that, in practice, this approach can handle most or all

the problems which are at the reach of the other approaches, with performances which are comparable with, or even better than, those of the current state-of-the-art tools.

Related work. First, we briefly overview the most successful approaches for the satisfiability of modal logics, and of K_m in particular. The "classic" *tableau-based* approach [5, 17, 10, 18] is based on the construction of propositional tableau branches, which are recursively expanded on demand by generating successor nodes in a candidate Kripke model. (Tools based on this approach are no more state-of-the-art.) In the *SAT-based* approach [7, 8] a DPLL procedure, which treats the modal subformulas as propositions, is used as boolean engine at each nesting level of the modal operators: when a satisfying assignment is found, the corresponding set of modal subformulas is recursively checked for modal consistency. Among the tools employing (and extending) this approach, we recall KSAT [7, 6], *SAT [25], FaCT and DLP [12], and RACER [26]. [1] In the *translational* approach [15, 1] the modal formula is encoded into first-order logic (FOL), and the encoded formula can be decided efficiently by a FOL theorem prover [1]. MSPASS [14] is the most representative tool of this approach. In the *Automata-theoretic* approach, (a BDD-based symbolic representation of) a tree automaton accepting all the tree models of the input formula is implicitly built and checked for emptiness [19, 20]. KBDD [20] is the representative tool of this approach. [20] presents also an encoding of K-satisfiability into QBF-satisfiability (that is a PSPACE-complete problem too), combined with the use of a state-of-the-art QBF solver.

Moreover, we briefly recall some approaches based on SAT encoding which have been successful in other domains: *planning* has been fruitfully encoded into SAT [16], and a similar encoding has been proposed for the problem of (bounded) LTL model checking [2]; in both cases, these approaches are currently state-of-the-art in the respective communities. Effective encodings into SAT have been proposed also for the satisfiability problems in some quantifier-free FOL theories which are of interest for formal verification, including these of *separation logic (SL)* [24], of *SL with equality and uninterpreted functions* [22], and of *linear arithmetic* [23].

Structure of the paper. In §2 we provide the necessary background notions. In §3 we describe the encoding, and provide some examples. In §4 we present an extensive empirical evaluation, and discuss the results. In §5 we conclude, an describe some possible future evolutions.

2 Background

We recall some basic definitions and properties of K_m. Given a non-empty set of primitive propositions $\mathcal{A} = \{A_1, A_2, \ldots\}$ and a set of m modal operators $\mathcal{B} = \{\Box_1, \ldots, \Box_m\}$, the language of K_m is the least set of formulas containing $\mathcal{A}$, closed under the set of propositional connectives $\{\neg, \wedge\}$ and the set of modal operators in $\mathcal{B}$. Notationally, we use the Greek letters $\alpha, \beta, \varphi, \psi, \nu, \pi$ to denote formulas in the language of K_m (K_m-formulas hereafter). We use the standard abbreviations, that is: "$\Diamond_r\varphi$" for "$\neg\Box_r\neg\varphi$", "$\varphi_1 \vee \varphi_2$" for "$\neg(\neg\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \rightarrow \varphi_2$" for "$\neg(\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \leftrightarrow \varphi_2$" for

[1] Notice that, for historical reasons, tools like FaCT, DLP, and RACER are called "tableau", although they are based on a DPLL engine and implement variants of the SAT-based schema.

"$\neg(\varphi_1 \wedge \neg\varphi_2) \wedge \neg(\varphi_2 \wedge \neg\varphi_1)$", "$\top$" and "$\bot$" for the constants "true" and "false". (Hereafter formulas like $\neg\neg\psi$ are implicitly assumed to be simplified into ψ, so that, if ψ is $\neg\phi$, then by "$\neg\psi$" we mean "ϕ".) We call *depth* of φ, written *depth*(φ), the maximum number of nested modal operators in φ. We call a *propositional atom* every primitive proposition in $\mathcal{A}$, and a *propositional literal* every propositional atom (*positive literal*) or its negation (*negative literal*).

In order to make our presentation more uniform, we adopt from [5, 18] the representation of K_m-formulas from the following table:

α	α_1	α_2	β	β_1	β_2	π^r	π^r_0	ν^r	ν^r_0
$(\varphi_1 \wedge \varphi_2)$	φ_1	φ_2	$(\varphi_1 \vee \varphi_2)$	φ_1	φ_2	$\Diamond_r\varphi_1$	φ_1	$\Box_r\varphi_1$	φ_1
$\neg(\varphi_1 \vee \varphi_2)$	$\neg\varphi_1$	$\neg\varphi_2$	$\neg(\varphi_1 \wedge \varphi_2)$	$\neg\varphi_1$	$\neg\varphi_2$	$\neg\Box_r\varphi_1$	$\neg\varphi_1$	$\neg\Diamond_r\varphi_1$	$\neg\varphi_1$
$\neg(\varphi_1 \rightarrow \varphi_2)$	φ_1	$\neg\varphi_2$	$(\varphi_1 \rightarrow \varphi_2)$	$\neg\varphi_1$	φ_2				

in which non-literal K_m-formulas are grouped into four categories: α's (conjunctive), β's (disjunctive), π's (existential), ν's (universal).

A *Kripke structure* for K_m is a tuple $\mathcal{M} = \langle \mathcal{U}, \mathcal{L}, \mathcal{R}_1, \ldots, \mathcal{R}_m \rangle$, where $\mathcal{U}$ is a set of states, $\mathcal{L}$ is a function $\mathcal{L} : \mathcal{A} \times \mathcal{U} \longmapsto \{True, False\}$, and each $\mathcal{R}_x$ is a binary relation on the states of $\mathcal{U}$. With an abuse of notation we write "$u \in \mathcal{M}$" instead of "$u \in \mathcal{U}$". We call a *situation* any pair $\mathcal{M}, u$, $\mathcal{M}$ being a Kripke structure and $u \in \mathcal{M}$. The binary relation $\models$ between a modal formula φ and a situation $\mathcal{M}, u$ is defined as follows:

$\mathcal{M}, u \models A_i,\ A_i \in \mathcal{A} \iff \mathcal{L}(A_i, u) = True$;
$\mathcal{M}, u \models \neg A_i,\ A_i \in \mathcal{A} \iff \mathcal{L}(A_i, u) = False$;
$\mathcal{M}, u \models \alpha \iff \mathcal{M}, u \models \alpha_1$ *and* $\mathcal{M}, u \models \alpha_2$;
$\mathcal{M}, u \models \beta \iff \mathcal{M}, u \models \beta_1$ *or* $\mathcal{M}, u \models \beta_2$;
$\mathcal{M}, u \models \pi^r \iff \mathcal{M}, w \models \pi^r_0$ for some $w \in \mathcal{U}$ s.t. $\mathcal{R}_x(u, w)$ holds in $\mathcal{M}$;
$\mathcal{M}, u \models \nu^r \iff \mathcal{M}, w \models \nu^r_0$ for every $w \in \mathcal{U}$ s.t. $\mathcal{R}_x(u, w)$ holds in $\mathcal{M}$.

"$\mathcal{M}, u \models \varphi$" should be read as "$\mathcal{M}, u$ *satisfy* φ in K_m" (alternatively, "$\mathcal{M}, u$ K_m-satisfies φ"). We say that a K_m-formula φ is satisfiable in K_m (K_m-satisfiable from now on) if and only if there exist $\mathcal{M}$ and $u \in M$ s.t. $\mathcal{M}, u \models \varphi$. (When this causes no ambiguity, we sometimes drop the prefix "K_m-".) We say that w is a *successor* of u through $\mathcal{R}_x$ iff $\mathcal{R}_x(u, w)$ holds in $\mathcal{M}$.

The problem of determining the K_m-satisfiability of a K_m-formula φ is decidable and PSPACE-complete [17, 10], even restricting the language to a single boolean atom (i.e., $\mathcal{A} = \{A_1\}$) [9]; if we impose a bound on the modal depth of the K_m-formulas, the problem reduces to NP-complete [9]. For a more detailed description on K_m— including, e.g., axiomatic characterization, decidability and complexity results — see [10, 9].

A K_m-formula is said to be in *Negative Normal Form (NNF)* if it is written in terms of the symbols $\Box_r$, $\Diamond_r$, $\wedge$, $\vee$ and propositional literals A_i, $\neg A_i$ (i.e., if all negations occur only before propositional atoms in $\mathcal{A}$). Every K_m-formula φ can be converted into an equivalent one $NNF(\varphi)$ by recursively applying the rewriting rules: $\neg\Box_r\varphi \Longrightarrow \Diamond_r\neg\varphi$, $\neg\Diamond_r\varphi \Longrightarrow \Box_r\neg\varphi$, $\neg(\varphi_1 \wedge \varphi_2) \Longrightarrow (\neg\varphi_1 \vee \neg\varphi_2)$, $\neg(\varphi_1 \vee \varphi_2) \Longrightarrow (\neg\varphi_1 \wedge \neg\varphi_2)$, $\neg\neg\varphi \Longrightarrow \varphi$.

A K_m-formula is said to be in *Box Normal Form (BNF)* [19, 20] if it is written in terms of the symbols $\Box_r$, $\neg\Box_r$, $\wedge$, $\vee$, and propositional literals A_i, $\neg A_i$ (i.e., if no diamonds are there, and all negations occurs only before boxes or before propositional atoms in $\mathcal{A}$). Every K_m-formula φ can be converted into an equivalent one $BNF(\varphi)$ by

recursively applying the rewriting rules: $\Diamond_r\varphi \Longrightarrow \neg\Box_r\neg\varphi$, $\neg(\varphi_1 \wedge \varphi_2) \Longrightarrow (\neg\varphi_1 \vee \neg\varphi_2)$, $\neg(\varphi_1 \vee \varphi_2) \Longrightarrow (\neg\varphi_1 \wedge \neg\varphi_2)$, $\neg\neg\varphi \Longrightarrow \varphi$.

3 The Encoding

We borrow some notation from the *Single Step Tableau (SST)* framework [18, 4]. We represent univocally states in $\mathcal{M}$ as labels σ, represented as non empty sequences of integers $1.n_1^{r_1}.n_2^{r_2}.\ ...\ .n_k^{r_k}$, s.t. the label 1 represents the root state, and $\sigma.n^r$ represents the n-th successor of σ through the relation $\mathcal{R}_r$. With a little abuse of notation, hereafter we may say "a state σ" meaning "a state labeled by σ".

Notationally, we often write "$(\bigwedge_i l_i) \rightarrow \bigvee_j l_j$" for the clause "$\bigvee_j \neg l_i \vee \bigvee_j l_j$", and "$(\bigwedge_i l_i) \rightarrow (\bigwedge_j l_j)$" for the conjunction of clauses "$\bigwedge_j (\bigvee_i \neg l_i \vee l_j)$".

3.1 The Basic Encoding

Let $A_{[,\,]}$ be an *injective* function which maps a pair $\langle\sigma, \psi\rangle$, s.t. σ is a state label and ψ is a K_m-formula which is not in the form $\neg\phi$, into a boolean variable $A_{[\sigma,\,\psi]}$. Let $L_{[\sigma,\,\psi]}$ denote $\neg A_{[\sigma,\,\phi]}$ if ψ is in the form $\neg\phi$, $A_{[\sigma,\,\psi]}$ otherwise. Given a K_m-formula φ, the encoder K_m2SAT builds a boolean CNF formula as follows:

$$K_m2SAT(\varphi) := A_{[1,\,\varphi]} \wedge Def(1,\,\varphi), \tag{1}$$

$$Def(\sigma,\,A_i), := \top \tag{2}$$

$$Def(\sigma,\,\neg A_i) := \top \tag{3}$$

$$Def(\sigma,\,\alpha) := (L_{[\sigma,\,\alpha]} \rightarrow (L_{[\sigma,\,\alpha_1]} \wedge L_{[\sigma,\,\alpha_2]})) \wedge Def(\sigma,\,\alpha_1) \wedge Def(\sigma,\,\alpha_2) \tag{4}$$

$$Def(\sigma,\,\beta) := (L_{[\sigma,\,\beta]} \rightarrow (L_{[\sigma,\,\beta_1]} \vee L_{[\sigma,\,\beta_2]})) \wedge Def(\sigma,\,\beta_1) \wedge Def(\sigma,\,\beta_2) \tag{5}$$

$$Def(\sigma,\,\pi^{r,j}) := (L_{[\sigma,\,\pi^{r,j}]} \rightarrow L_{[\sigma.j,\,\pi_0^{r,j}]}) \wedge Def(\sigma.j,\,\pi_0^{r,j}) \tag{6}$$

$$Def(\sigma,\,\nu^r) := \bigwedge_{\langle\sigma:\pi^{r,i}\rangle} ((L_{[\sigma,\,\nu^r]} \wedge L_{[\sigma,\,\pi^{r,i}]}) \rightarrow L_{[\sigma.i,\,\nu_0^r]}) \ \wedge \bigwedge_{\langle\sigma:\pi^{r,i}\rangle} Def(\sigma.i,\,\nu_0^r). \tag{7}$$

We assume that the K_m-formulas are represented as DAGs, so that to avoid the expansion of the same $Def(\sigma,\,\psi)$ more than once. Moreover, following [18], we assume that, for each σ, the $Def(\sigma,\,\psi)$'s are expanded in the order: α, β, π, ν. Thus, each $Def(\sigma,\,\nu^r)$ is expanded after the expansion of all $Def(\sigma,\,\pi^{r,i})$'s, so that $Def(\sigma,\,\nu^r)$ will generate one clause $((L_{[\sigma,\,\pi^{r,i}]} \wedge L_{[\sigma,\,\Box_r\nu_0^r]}) \rightarrow L_{[\sigma.i,\,\nu_0^r]})$ and one novel definition $Def(\sigma.i,\,\nu_0^r)$ for each $Def(\sigma,\,\pi^{r,i})$ expanded.

Intuitively, it is easy to see that $K_m2SAT(\varphi)$ mimics the construction of an SST tableau expansion [18, 4]. We have the following fact.

Theorem 1. *A K_m-formula φ is K_m-satisfiable if and only if the corresponding boolean formula $K_m2SAT(\varphi)$ is satisfiable.*

The complete proof of Theorem 1 can be found in the Appendix.

Notice that, due to (7), the number of variables and clauses in $K_m2SAT(\varphi)$ may grow exponentially with $depth(\varphi)$. This is in accordance to what stated in [9].

3.2 Variants

Before the encoding, some potentially useful preprocessing can be performed.

First, the input K_m-formulas can be converted into NNF (like, e.g., in [18, 4]) or into BNF (like, e.g., in [7, 19]). One potential advantage of the latter is that, when one $\Box_r\psi$ occurs both positively and negatively (like, e.g., in $(\Box_r\psi \vee ...) \wedge (\neg\Box_r\psi \vee ...) \wedge ...$), then both occurrences of $\Box_r\psi$ are labeled by the same boolean atom $A_{[\sigma,\ \Box_r\psi]}$, and hence they are always assigned the same truth value by DPLL; with NNF, instead, the negative occurrence $\neg\Box_r\psi$ is rewritten into $\Diamond_r\neg\psi$, so that two distinct boolean atoms $A_{[\sigma,\ \Box_r\psi]}$ and $A_{[\sigma,\ \Diamond_r\neg\psi]}$ are generated; DPLL can assign them the same truth value, creating a hidden conflict which may require some extra boolean search to reveal.

Example 1 (NNF). Let φ_{nnf} be $(\Diamond A_1 \vee \Diamond(A_2 \vee A_3)) \wedge \Box\neg A_1 \wedge \Box\neg A_2 \wedge \Box\neg A_3$.[2] It is easy to see that φ_{nnf} is K_1-unsatisfiable. $K_m2SAT(\varphi_{nnf})$ is:

$$
\begin{array}{lll}
1. & & A_{[1,\ \varphi_{nnf}]} \\
2. & \wedge & \left(A_{[1,\ \varphi_{nnf}]} \rightarrow (A_{[1,\ \Diamond A_1 \vee \Diamond(A_2 \vee A_3)]} \wedge A_{[1,\ \Box\neg A_1]} \wedge A_{[1,\ \Box\neg A_2]} \wedge A_{[1,\ \Box\neg A_3]}) \right) \\
3. & \wedge & \left(A_{[1,\ \Diamond A_1 \vee \Diamond(A_2 \vee A_3)]} \rightarrow (A_{[1,\ \Diamond A_1]} \vee A_{[1,\ \Diamond(A_2 \vee A_3)]}) \right) \\
4. & \wedge & \left(A_{[1,\ \Diamond A_1]} \rightarrow A_{[1.1,\ A_1]} \right) \\
5. & \wedge & \left(A_{[1,\ \Diamond(A_2 \vee A_3)]} \rightarrow A_{[1.2,\ A_2 \vee A_3]} \right) \\
6. & \wedge & \left((A_{[1,\ \Box\neg A_1]} \wedge A_{[1,\ \Diamond A_1]}) \rightarrow \neg A_{[1.1,\ A_1]} \right) \\
7. & \wedge & \left((A_{[1,\ \Box\neg A_2]} \wedge A_{[1,\ \Diamond A_1]}) \rightarrow \neg A_{[1.1,\ A_2]} \right) \\
8. & \wedge & \left((A_{[1,\ \Box\neg A_3]} \wedge A_{[1,\ \Diamond A_1]}) \rightarrow \neg A_{[1.1,\ A_3]} \right) \\
9. & \wedge & \left((A_{[1,\ \Box\neg A_1]} \wedge A_{[1,\ \Diamond(A_2 \vee A_3)]}) \rightarrow \neg A_{[1.2,\ A_1]} \right) \\
10. & \wedge & \left((A_{[1,\ \Box\neg A_2]} \wedge A_{[1,\ \Diamond(A_2 \vee A_3)]}) \rightarrow \neg A_{[1.2,\ A_2]} \right) \\
11. & \wedge & \left((A_{[1,\ \Box\neg A_3]} \wedge A_{[1,\ \Diamond(A_2 \vee A_3)]}) \rightarrow \neg A_{[1.2,\ A_3]} \right) \\
12. & \wedge & \left(A_{[1.2,\ A_2 \vee A_3]} \rightarrow (A_{[1.2,\ A_2]} \vee A_{[1.2,\ A_3]}) \right)
\end{array}
$$

After a run of BCP, 3. reduces to a disjunction. If the first element $A_{[1,\ \Diamond A_1]}$ is assigned, then by BCP we have a conflict on 4.,6. If the second element $A_{[1,\ \Diamond(A_2 \vee A_3)]}$ is assigned, then by BCP we have a conflict on 12. Thus $K_m2SAT(\varphi_{nnf})$ is unsatisfiable. $\diamond$

Example 2 (BNF). Let $\varphi_{bnf} = (\neg\Box\neg A_1 \vee \neg\Box(\neg A_2 \wedge \neg A_3)) \wedge \Box\neg A_1 \wedge \Box\neg A_2 \wedge \Box\neg A_3$. It is easy to see that φ_{bnf} is K_1-unsatisfiable. $K_m2SAT(\varphi_{bnf})$ is: [3]

[2] For K_1 formulas, we omit the box and diamond indexes.

[3] Notice that the valid clause 6. can be omitted.

$$
\begin{array}{lll}
1. & & A_{[1,\ \varphi_{bnf}]} \\
2. & \wedge & \left(A_{[1,\ \varphi_{bnf}]} \rightarrow \left(A_{[1,\ (\neg\Box\neg A_1 \vee \neg\Box(\neg A_2 \wedge \neg A_3))]} \wedge A_{[1,\ \Box\neg A_1]} \wedge A_{[1,\ \Box\neg A_2]} \wedge A_{[1,\ \Box\neg A_3]}\right)\right) \\
3. & \wedge & \left(A_{[1,\ (\neg\Box\neg A_1 \vee \neg\Box(\neg A_2 \wedge \neg A_3))]} \rightarrow \left(\neg A_{[1,\ \Box\neg A_1]} \vee \neg A_{[1,\ \Box(\neg A_2 \wedge \neg A_3)]}\right)\right) \\
4. & \wedge & \left(\neg A_{[1,\ \Box\neg A_1]} \rightarrow A_{[1.1,\ A_1]} \right) \\
5. & \wedge & \left(\neg A_{[1,\ \Box(\neg A_2 \wedge \neg A_3)]} \rightarrow \neg A_{[1.2,\ (\neg A_2 \wedge \neg A_3)]} \right) \\
6. & \wedge & \left(\left(A_{[1,\ \Box\neg A_1]} \wedge \neg A_{[1,\ \Box\neg A_1]}\right) \rightarrow \neg A_{[1.1,\ A_1]} \right) \\
7. & \wedge & \left(\left(A_{[1,\ \Box\neg A_2]} \wedge \neg A_{[1,\ \Box\neg A_1]}\right) \rightarrow \neg A_{[1.1,\ A_2]} \right) \\
8. & \wedge & \left(\left(A_{[1,\ \Box\neg A_3]} \wedge \neg A_{[1,\ \Box\neg A_1]}\right) \rightarrow \neg A_{[1.1,\ A_3]} \right) \\
9. & \wedge & \left(\left(A_{[1,\ \Box\neg A_1]} \wedge \neg A_{[1,\ \Box(\neg A_2 \wedge \neg A_3)]}\right) \rightarrow \neg A_{[1.2,\ A_1]} \right) \\
10. & \wedge & \left(\left(A_{[1,\ \Box\neg A_2]} \wedge \neg A_{[1,\ \Box(\neg A_2 \wedge \neg A_3)]}\right) \rightarrow \neg A_{[1.2,\ A_2]} \right) \\
11. & \wedge & \left(\left(A_{[1,\ \Box\neg A_3]} \wedge \neg A_{[1,\ \Box(\neg A_2 \wedge \neg A_3)]}\right) \rightarrow \neg A_{[1.2,\ A_3]} \right) \\
12. & \wedge & \left(\neg A_{[1.2,\ (\neg A_2 \wedge \neg A_3)]} \rightarrow \left(A_{[1.2,\ A_2]} \vee A_{[1.2,\ A_3]}\right) \right)
\end{array}
$$

Unlike with NNF, $K_m2SAT(\varphi_{bnf})$ is found unsatisfiable directly by BCP. Notice that the unit-propagation of $A_{[1,\ \Box\neg A_1]}$ from 2. causes $\neg A_{[1,\ \Box\neg A_1]}$ in 3. to be false, so that one of the two (unsatisfiable) branches induced by the disjunction is cut a priori. With NNF, the corresponding atoms $A_{[1,\ \Box\neg A_1]}$ and $A_{[1,\ \Diamond A_1]}$ are not recognized to be one the negation of the other, s.t. DPLL may need exploring one boolean branch more. $\diamond$

Second, the (NNF or BNF) K_m-formula can also be rewritten by recursively applying the validity-preserving "box/diamond lifting rules":

$$(\Box_r\varphi_1 \wedge \Box_r\varphi_2) \Longrightarrow \Box_r(\varphi_1 \wedge \varphi_2), \quad (\Diamond_r\varphi_1 \vee \Diamond_r\varphi_2) \Longrightarrow \Diamond_r(\varphi_1 \vee \varphi_2). \tag{8}$$

This has the potential benefit of reducing the number of $\pi^{r,i}$ formulas, and hence the number of labels $\sigma.i$ to take into account in the expansion of the $Def(\sigma,\ \nu^r)$'s (7).

Example 3 (NNF with LIFT). Let $\varphi_{nnflift} = \Diamond(A_1 \vee A_2 \vee A_3) \wedge \Box(\neg A_1 \wedge \neg A_2 \wedge \neg A_3)$. It is easy to see that $\varphi_{bnflift}$ is K_1-unsatisfiable. $K_m2SAT(\varphi_{bnflift})$ is:

$$
\begin{array}{lll}
 & & A_{[1,\ \varphi_{nnflift}]} \\
\wedge & & \left(A_{[1,\ \varphi_{nnflift}]} \rightarrow \left(A_{[1,\ \Diamond(A_1 \vee A_2 \vee A_3)]} \wedge A_{[1,\ \Box(\neg A_1 \wedge \neg A_2 \wedge \neg A_3)]}\right)\right) \\
\wedge & & \left(A_{[1,\ \Diamond(A_1 \vee A_2 \vee A_3)]} \rightarrow A_{[1.1,\ A_1 \vee A_2 \vee A_3]} \right) \\
\wedge & & \left(\left(A_{[1,\ \Box(\neg A_1 \wedge \neg A_2 \wedge \neg A_3)]} \wedge A_{[1,\ \Diamond(A_1 \vee A_2 \vee A_3)]}\right) \rightarrow A_{[1.1,\ (\neg A_1 \wedge \neg A_2 \wedge \neg A_3)]} \right) \\
\wedge & & \left(A_{[1.1,\ A_1 \vee A_2 \vee A_3]} \rightarrow \left(A_{[1.1,\ A_1]} \vee A_{[1.1,\ A_2]} \vee A_{[1.1,\ A_3]}\right) \right) \\
\wedge & & \left(A_{[1.1,\ (\neg A_1 \wedge \neg A_2 \wedge \neg A_3)]} \rightarrow \left(\neg A_{[1.1,\ A_1]} \wedge \neg A_{[1.1,\ A_2]} \wedge \neg A_{[1.1,\ A_3]}\right) \right)
\end{array}
$$

$K_m2SAT(\varphi_{nnflift})$ is found unsatisfiable by BCP. $\diamond$

Example 4 (BNF with LIFT). Let $\varphi_{bnflift} = \neg\Box(\neg A_1 \wedge \neg A_2 \wedge \neg A_3) \wedge \Box(\neg A_1 \wedge \neg A_2 \wedge \neg A_3)$. It is easy to see that $\varphi_{bnflift}$ is K_1-unsatisfiable. $K_m2SAT(\varphi_{bnflift})$ is:

$$
\begin{array}{lll}
1. & & A_{[1,\ \varphi_{bnflift}]} \\
2. & \wedge & \left(A_{[1,\ \varphi_{bnflift}]} \rightarrow \left(\neg A_{[1,\ \Box(\neg A_1 \wedge \neg A_2 \wedge \neg A_3)]} \wedge A_{[1,\ \Box(\neg A_1 \wedge \neg A_2 \wedge \neg A_3)]}\right)\right) \\
3. & \wedge & \left(\neg A_{[1,\ \Box(\neg A_1 \wedge \neg A_2 \wedge \neg A_3)]} \rightarrow \neg A_{[1.1,\ (\neg A_1 \wedge \neg A_2 \wedge \neg A_3)]} \right) \\
4. & \wedge & \left(\neg A_{[1.1,\ (\neg A_1 \wedge \neg A_2 \wedge \neg A_3)]} \rightarrow \left(A_{[1.1,\ A_1]} \vee A_{[1.1,\ A_2]} \vee A_{[1.1,\ A_3]}\right) \right)
\end{array}
$$

$K_m2SAT(\varphi_{bnflift})$ is found unsatisfiable by BCP. $\diamond$

One potential drawback of applying the lifting rules is that, by collapsing $(\Box_r\varphi_1 \wedge \Box_r\varphi_2)$ into $\Box_r(\varphi_1 \wedge \varphi_2)$ and $(\Diamond_r\varphi_1 \vee \Diamond_r\varphi_2)$ into $\Diamond_r(\varphi_1 \vee \varphi_2)$, the possibility of sharing box/diamond subformulas in the DAG representation of φ is reduced.

4 Empirical Evaluation

In order to verify empirically the effectiveness of this approach, we have performed an extensive empirical test session. We have implemented the encoder $K_m 2SAT$ in C++, with two flags: `NNF`/`BNF`, performing a pre-conversion into NNF/BNF before the encoding, and `lift`/`nolift`, performing box lifting before the encoding. We have tried many SAT solvers on our encoded formulas (including ZCHAFF `2004.11.15`, SIEGE `v4`, BERKMIN `5.6.1`, MINISAT `v1.13`, SAT-ELITE `v1.0` and SAT-ELITE GTI `2005 submission`).). After a preliminary evaluation, we have selected ZCHAFF and SAT-ELITE GTI (hereafter SAT-ELITE).

Among the state-of-the-art tools for K_m-satisfiability, we have selected RACER [26] and *SAT [25] as the best representers of the tableaux/DPLL-based tools, MSPASS [15, 14] as the best representer of the FOL-encoding approach, [4] KBDD [19, 20] [5] as the representer of the automata-theoretic approach, and the K-QBF translator [20] combined with the SEMPROP QBF solver (which seems to be the best QBF solver for these kinds of problems [6]) as representant of the QBF-encoding approach. [7]

All tests presented in this section have been performed on a two-processor Intel Xeon 3.0GHz computer, with 1 MByte Cache each processor, 4 GByte RAM, with Red Hat Linux 3.0 Enterprise Server, where four processes can run in parallel. When reporting the results for one $K_m 2SAT$ +DPLL version, the CPU times reported are the sums of both the encoding and DPLL solving times.

In order to make the results reproducible, the encoder, the benchmarks and the files with all the plots are available. [8]

4.1 Test description

As a first group of benchmark formulas we used the LWB benchmark suite used in a comparison at Tableaux'98 [11]. It consists on 9 classes of parameterized formulas (each in two versions, valid "_p" or invalid "_n"), for a total amount of 378 formulas. The parameter allows for creating formulas of increasing size and difficulty.

The benchmark methodology is to test formulas from each class, in increasing difficulty, until one formula cannot be solved within a given timeout (1000 seconds in our tests)[9]. The result from this class is the parameter's value of the largest (and hardest) formula that can be solved within the time limit. (The parameter ranges only from 1 to

[4] We have run MSPASS with the options `-EMLTranslation=2 -EMLFuncNary=1 -Sorts=0 -CNFOptSkolem=0 -CNFStrSkolem=0 -Select=2 -Split=-1 -DocProof=0 -PProblem=0 -PKept=0 -PGiven=0`, which are suggested for K_m-formulas in the MSPASS README file.

[5] KBDD has been run with the default internal memory bound of 384MB.

[6] In results we report only SEMPROP times, since the time spent by K-QBF is almost negligiable.

[7] Other tools like LEANK, □KE, LWB, KRIS are not competitive with the ones listed above [13]. Tools like DLP and FACT are similar in spirit and performances to RACER and *SAT. KSAT [7, 8, 6] has been reimplemented into *SAT.

[8] Available at `http://www.dit.unitn.it/~rseba/sat06/allmaterial.tar.gz`.

[9] We also set a 1GB file-size limit for the encoding produced by $K_m 2SAT$.

21 so that, if a system can solve all 21 instances of a class, the result is given as 21.) For a discussion on this benchmark suite, we refer the reader to [11, 13].

As a second group of benchmark formulas, we have selected the random $\Box_m$-CNF testbed described in [13, 21]. This is a generalization of the well-known random k-SAT test methods, and is the final result on a long discussion in the communities of modal and description logics on how to to obtain significant and flawless random benchmarks for modal/description logics (see [7, 15, 6, 13, 21]).

In the $\Box_m$-CNF test methodology, a $\Box_m$-CNF formula is randomly generated according to the following parameters:

- the (maximum) modal depth d;
- the number of top-level clauses L;
- the number of literal per clause clauses k;
- the number of distinct propositional variables N;
- the number of distinct box symbols m;
- the percentage p of purely propositional literals in clauses occurring at depth $< d$. [10]

(We refer the reader to [13, 21] for a more detailed description.)

A typical problem set is characterized by fixed values of d, k, N, m, and p: L is varied in such a way as to empirically cover the "100% satisfiable—100% unsatisfiable" transition. Then, for each tuple of the five values in a problem set, a certain number of $\Box_m$-CNF formulas are randomly generated, and the resulting formulas are given in input to the procedure under test, with a maximum time bound. The fraction of formulas which were solved within a given timeout, and the median/percentile values of CPU times are plotted against the ratio L/N. Also, the fraction of satisfiable/unsatisfiable formulas is plotted for a better understanding.

Following [13, 21], we have fixed $m = 1, k = 3$ and 100 samples/point in all our tests, and we have selected two groups: $d = 1$, $p = 0.5$, and $N = 6, 7, 8, 9$, $L/N = 10..60$, and $d = 2$, $p = 0.5, 0.6$, $N = 3, 4$, $L/N = 30..150$, for a total amount of 12,000 formulas. In each test, we imposed a timeout of 500 seconds per sample [11] and we plot the number of samples which were solved within the timeout, and the 50%th and 90%th percentiles of CPU time. In order to correlate the performances with the (un)satisfiability of the sample formulas, in the background of each plot we plot the satisfiable/unsatisfiable ratio.

For both benchmark suites, for all formulas, all tools under test —both all the variants of $K_m 2SAT$ +DPLL and all the state-of-the-art K_m-satisfiability solvers— agreed on the satisfiability/unsatisfiability result when terminating within the timeout.

4.2 An empirical comparison of the different variants of $K_m 2SAT$

We have evaluated the four variants of the encoding, with both ZCHAFF and SATELITE.

[10] Each modal clause of length k contains on average $p \cdot k$ randomly-picked boolean literals and $k - p \cdot k$ randomly-generated modal literals $\Box_r \psi$, $\neg \Box_r \psi$. More precisely, the number of boolean literals in a clause is $\lfloor p \cdot k \rfloor$ (resp. $\lceil p \cdot k \rceil$) with probability $\lceil p \cdot k \rceil - p \cdot k$ (resp. $1 - (\lceil p \cdot k \rceil - p \cdot k)$). Notice that typically the smaller is p, the harder is the problem [13, 21].

[11] With also a 512MB file-size limit for the encoding produced by $K_m 2SAT$

Encoding	number of variables ($\cdot 10^3$)				number of clauses ($\cdot 10^3$)				ZCHAFF				SAT-ELITE			
	NNF		BNF		NNF		BNF		NNF		BNF		NNF		BNF	
Lifting	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no
k_branch_n	1000	1000	1000	1000	1000	1000	1000	1000	4	4	4	4	4	4	4	4
k_branch_p	1000	1000	1000	1000	1000	1000	1000	1000	4	4	4	4	4	4	4	4
k_d4_n	500	2000	2000	1000	600	2000	3000	2000	4	4	6	6	5	6	5	5
k_d4_p	5000	5000	9000	9000	6000	6000	11000	11000	9	9	9	9	9	9	10	10
k_dum_n	5000	5000	5000	5000	6000	6000	6000	6000	18	18	18	18	18	18	18	17
k_dum_p	7000	7000	5000	6000	8000	9000	7000	7000	17	17	17	17	16	16	16	15
k_grz_n	10	10	10	10	10	10	10	10	21	21	21	21	21	21	21	21
k_grz_p	10	10	9	8	10	10	9	8	21	21	21	21	21	21	21	21
k_lin_n	30	30	30	30	50	50	500	50	21	21	21	21	21	21	21	21
k_lin_p	4	10	4	10	5	10	5	10	21	21	21	21	21	21	21	21
k_path_n	2000	2000	2000	2000	2000	2000	2000	2000	5	5	5	6	6	6	6	6
k_path_p	2000	12000	2000	11000	2000	14000	2000	13000	7	7	7	7	7	8	7	8
k_ph_n	300	50	300	50	300	50	300	50	21	21	21	21	21	21	21	21
k_ph_p	10	4	10	5	10	5	10	6	11	11	11	12	11	11	10	11
k_poly_n	20	200	20	200	20	200	20	200	21	21	21	21	21	21	21	21
k_poly_p	20	200	20	200	20	200	20	200	21	21	21	21	21	21	21	21
k_t4p_n	800	1000	3000	4000	900	1000	3000	4000	4	4	5	5	4	4	5	4
k_t4p_p	3000	3000	4000	4000	3000	4000	4000	5000	8	8	9	9	8	8	9	9

Table 1. Comparison of the variants of K_m2SAT +ZCHAFF/SAT-ELITE on the LWB benchmarks.

The results on the LWB benchmark are summarized in Table 1. The first block reports the number of variables and clauses of $K_m2SAT(\varphi)$ (referring to the highest case solved). The second block reports the maximum parameter's value of the hardest formula solved within the timeout. (E.g., the `BNF-nolifting` encoding of `k_d4_p(10)` contains $9 \cdot 10^6$ variables and $11 \cdot 10^6$ clauses, it is the hardest `k_d4_p` problem solved by SAT-ELITE, and it is the first which is not solved by ZCHAFF.)

Looking at the number of clauses and variables, we notice a few facts: first, `lifting` in many cases reduces the size of the encoded formula (e.g., in `k_path_p`), with a few exceptions (e.g., in `k_grz_p`); second, `BNF` in many cases produces smaller formulas than `NNF` (e.g., in `k_dum_p`), with a few exceptions (e.g., in `k_ph_p`).

Looking at the CPU times, we notice that the performance gaps are rather small, and there is no absolute winner neither between the SAT solvers nor among the encodings, although the `BNF-nolifting` version with ZCHAFF seems to behave slightly better than the others on the whole. We also notice that there seems not to be an absolute correlation between a reduction in size of the encoded formula and an improvement of performances. (E.g., with `k_path_p` and `BNF`, lifting reduces significantly the size of the formula, but it causes a worse performance of SAT-ELITE.)

In the random $\square_m$-CNF benchmarks the gaps are more relevant. (For better readability, we report only the results for ZCHAFF because it turned out to be significantly better then SAT-ELITE on these formula, with a noteworthy exception of the second row in Figure 4, where we reported also the values for SAT-ELITE.)

For lack of space, we do not present the plots of the number of variables and clauses of the encoded formulas. Anyway, we report a few facts: first, in all the tests, these values grow linearly (or slightly sub-linearly [12]) with the number of clauses L of the K_m-formula; second, the values are very regular: the gaps between 50% and 90% percentiles are negligible; third, `BNF` produces slightly smaller formulas than `NNF` in most cases, but the gaps are small; fourth, `lifting` increases significantly the number of variables and clauses generated. [13] To give a coarse idea of the values, with the hardest problems ($d = 2$, $p = 0.5$, $N = 4$, $L/N = 150$) and the worst encoder (`lift`, `NNF`), we have about $2.5 \cdot 10^6$ clauses and $1.5 \cdot 10^6$ variables, whilst with the best encoder (`nolift`, `BNF`) we have about $1.2 \cdot 10^6$ clauses and $0.7 \cdot 10^6$ variables.

The CPU times are reported in Figures 1 and 2. The tests of Figure 1 are simply too easy for *K_m2SAT* +ZCHAFF (but not for its competitors, see later) which solved every sample formula in less than 1 second. The tests of Figure 2 are more challenging. In general, it seems that in the majority of cases `nolifting` beats `lifting` and `BNF` beats `NNF`, although this seems not to be a general rule. In the hardest benchmark with $p = 0.5$ and $N = 4$ (2nd row), we also reported the SAT-ELITE plots, because it beats significantly ZCHAFF.

Finally, we notice that for *K_m2SAT* +ZCHAFF the problems tend to be harder within the satisfiability/unsatisfiability transition area. This fact holds also for RACER and *SAT, see figures 1 and 2. This seems to confirm the fact that the easy-hard-easy pattern of random k-SAT extends also to $\Box_m$-CNF, as already observed in [7, 8, 6, 13, 21].

4.3 An empirical comparison wrt. other approaches

We have evaluated the *K_m2SAT* +DPLL approach against the other K_m-satisfiability solvers listed above.

The results on the LWB benchmark are summarized in Table 2. We notice a few facts: first, RACER and *SAT are the best scorers (confirming the analysis in [13]); second, there is no definite winner between KBDD and MSPASS; third, K-QBF +SEMPROP is the worst performer among the state-of-the-art tools, way below the performances of RACER and *SAT; fourth, *K_m2SAT* +DPLL is the worst performer on `k_branch`, `k_d4`, `k_dum` the second worst performer on `k_path`, `k_t4p`, it equals the competitors on `k_grz` and `k_lin`, and it is (one of) the best performer(s) on `k_ph` and `k_Pol`.

In the random $\Box_m$-CNF benchmarks the results are much better for *K_m2SAT*. In Figure 3 (center and right) *K_m2SAT* +ZCHAFF is nearly always the best scorer, followed by *SAT and RACER. From Figure 3 (left) notice that K-QBF +SEMPROP, MSPASS, and KBDD do not terminate within the timeout for most values, whilst *K_m2SAT* +ZCHAFF always terminates (below 1 second). In Figure 4 (center and right) we notice that, for $p = 0.5$ (1st and 2nd row) *K_m2SAT* +ZCHAFF is nearly always the best scorer (notice that with $N = 4$ SAT-ELITE beats ZCHAFF); for $p = 0.6$ (3rd and 4th row) *K_m2SAT* +ZCHAFF is beaten only by *SAT. In all these tests, K-QBF +SEMPROP, MSPASS and KBDD are not competitive.

[12] The more clauses are a in $\Box_m$-CNF formula, the higher are the chances of sharing subformulas.

[13] We believe that this may be due to the fact that random $\Box_m$-CNF formulas may contain lots of shared subformulas $\Box_r\psi$, so that lifting may cause a reduction of such sharing (see §3).

test	state-of-the-art tools					K_m2SAT best (BNF-nolift + ZCHAFF)	
	K-QBF + SEMPROP	KBDD	MSPASS	RACER	*SAT	solved	encoded
k_branch_n	10	8	10	15	14	4	4
k_branch_p	10	8	10	21	21	4	4
k_d4_n	9	21	21	21	21	6	7
k_d4_p	15	21	21	21	21	9	11
k_dum_n	21	21	21	21	21	18	20
k_dum_p	21	21	21	21	21	17	19
k_grz_n	11	21	21	21	21	21	21
k_grz_p	13	21	21	21	21	21	21
k_lin_n	21	21	21	21	21	21	21
k_lin_p	3	21	3	21	21	21	21
k_path_n	9	10	4	21	21	5	7
k_path_p	10	15	5	21	21	7	8
k_ph_n	8	4	12	21	13	21	21
k_ph_p	7	4	8	9	9	11	21
k_poly_n	21	8	7	21	21	21	21
k_poly_p	21	7	7	21	21	21	21
k_t4p_n	4	21	12	21	21	5	6
k_t4p_p	6	21	21	21	21	9	11

Table 2. Comparison of K_m2SAT +ZCHAFF againts the state-of-the-art tools on the LWB benchmarks. The righmost columns represents the biggest formula encoded within the 1GB bound.

4.4 Discussion

As highlighted in [6, 13], the satisfiability problem in modal logics like K_m is characterized by the alternation of two orthogonal components of reasoning: a *boolean* component, performing boolean reasoning within each state, and a *modal* component, generating the successor states of each state. The latter must cope with the fact that the candidate models may be up to exponentially big wrt. $depth(\varphi)$, whilst the former must cope with the fact that there may be up to exponentially many candidate (sub)models to explore. In the K_m2SAT +DPLL approach the encoder has to handle the whole modal component ((6) and (7)), whilst the handling of the whole boolean component is delegated to the SAT solver.

From the results displayed in this section we notice that, there are formulas for which the K_m2SAT +DPLL approach is much more inefficient than other state-of-the-art approaches (e.g., the `k_branch_n` formulas), and others for which it is much more efficient (e.g., the `k_ph_p` or the $\Box_m$-CNF formulas with $d = 1$).

On one extreme, the `k_branch_n` formulas are very hard from the perspective of modal reasoning, because they require finding one model $\mathcal{M}$ with $2^{d+1} - 1$ states [10] (but no boolean reasoning within each state is really required [6, 13] [14]). Thus, the size of the $K_m2SAT(\varphi)$ formulas blows up very quickly. On the other extreme, `k_ph_p(d)`

[14] A tool like *SAT solves `k_branch_n(d)` with $2^{d+1} - 1$ calls to its embedded DPLL engine, one for each state of $\mathcal{M}$, each call solved by BCP only.

is the modal encoding of a hard boolean problem [11], so that the modal component of reasoning is limited. Thus, the $K_m2SAT(\varphi)$ formulas have a reasonable size even for $d = 21$, although they are very hard boolean formulas even for $d = 12$ (anyway, ZCHAFF can handle them much better than other tools can handle the corresponding K_m-formulas).

On the whole, the K_m2SAT +DPLL approach is outperformed by other approaches on problems where the modal component of reasoning dominates (like, e.g., the `k_branch_n` formulas), and outperforms them on problems where the boolean component of reasoning dominates (like, e.g., the `k_ph_n` or the $\Box_m$-CNF formulas with $d = 1$), For formulas in which both components are relevant (e.g., the $\Box_m$-CNF formulas with $d = 2$ and $p = 0.5$, see [21]), the K_m2SAT +DPLL approach is competitive wrt. the other approaches, although no absolute winner can be established.

5 Conclusions and future work

In this paper we have explored the idea of encoding $K_m/\mathcal{ALC}$-satisfiability into SAT, so that to be handled by state-of-the-art SAT tools. We have showed that, despite the intrinsic risk of blowup in the size of the encoded formulas, the performances of this approach are comparable with those of current state-of-the-art tools on a rather extensive variety of empirical tests. (Notice that, as a byproduct of this work, the encoding of hard K_m-formulas could be used as benchmarks for SAT solvers.)

We see many possible direction to explore in order to enhance and extend this approach. First, our current implementation of the encoder is very straightforward, and optimizations for making the formula more compact can be introduced. Second, techniques implemented in other approaches (e.g., the pure literal optimization of [20]) could be imported. Third, hybrid approaches between K_m2SAT and KSAT-style tools could be investigated.

Another important open research line is to explore encodings for other modal and description logics. Whilst for logics like T_m the extension should be straightforward, logics like $S4_m$, or more elaborated description logics than $\mathcal{ALC}$, should be challenging.

References

1. C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-Based Heuristics in Modal Theorem Proving. In *Proc. ECAI'00*, 2000.
2. A. Biere, A. Cimatti, E. M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In *Proc. TACAS'99*, pages 193–207, 1999.
3. S. Brand, R. Gennari, and M. de Rijke. Constraint Programming for Modelling and Solving Modal Satisfability. In *Proc. CP 2003*, volume 3010 of *LNAI*. Springer, 2003.
4. F. Donini and F. Massacci. EXPTIME tableaux for ALC. *Artificial Intelligence*, 124(1):87–138, 2000.
5. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel Publishg, 1983.
6. E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. SAT vs. Translation based decision procedures for modal logics: a comparative evaluation. *Journal of Applied Non-Classical Logics*, 10(2), 2000.

7. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. CADE'13*. Springer, 1996.
8. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K(m). *Information and Computation*, 162(1/2), 2000.
9. J. Y. Halpern. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence*, 75(3):361–372, 1995.
10. J.Y. Halpern and Y. Moses. A guide to the completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, 1992.
11. A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. Technical Report IAM-96-015, University of Bern, Switzerland, 1996.
12. I. Horrocks and P. F. Patel-Schneider. Optimizing Description Logic Subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.
13. I. Horrocks, P. F. Patel-Schneider, and R. Sebastiani. An Analysis of Empirical Testing for Modal Decision Procedures. *Logic Journal of the IGPL*, 8(3), 2000.
14. U. Hustadt, R. A. Schmidt, and C. Weidenbach. MSPASS: Subsumption Testing with SPASS. In *Proc. DL'99*, pages 136–137, 1999.
15. U. Hustadt and R.A. Schmidt. An empirical analysis of modal theorem provers. *Journal of Applied Non-Classical Logics*, 9(4), 1999.
16. H. Kautz, D. McAllester, and B. Selman. Encoding Plans in Propositional Logic. In *Proceedings KR'96*. AAAI Press, 1996.
17. R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comp.*, 6(3):467–480, 1977.
18. F. Massacci. Single Step Tableaux for modal logics: methodology, computations, algorithms. *Journal of Automated Reasoning*, Vol. 24(3), 2000.
19. G. Pan, U. Sattler, and M. Y. Vardi. BDD-Based Decision Procedures for K. In *Proc. CADE*, LNAI. Springer, 2002.
20. G. Pan and M. Y. Vardi. Optimizing a BDD-based modal solver. In *Proc. CADE*, LNAI. Springer, 2003.
21. P. F. Patel-Schneider and R. Sebastiani. A New General Method to Generate Random Modal Formulae for Testing Decision Procedures. *Journal of Artificial Intelligence Research, (JAIR)*, 18, 2003. Morgan Kaufmann.
22. S. A. Seshia, S. K. Lahiri, and R. E. Bryant. A Hybrid SAT-Based Decision Procedure for Separation Logic with Uninterpreted Functions. In *Proc. DAC'03*, 2003.
23. O. Strichman. On Solving Presburger and Linear Arithmetic with SAT. In *Proc. of Formal Methods in Computer-Aided Design (FMCAD 2002)*, LNCS. Springer, 2002.
24. O. Strichman, S. Seshia, and R. Bryant. Deciding separation formulas with SAT. In *Proc. CAV'02*, LNCS. Springer, 2002.
25. Armando Tacchella. *SAT system description. In *Proc. DL'99*, pages 142–144, 1999.
26. R. Moeller V. Haarslev. RACER System Description. In *Proc. IJCAR-2001*, volume 2083 of *LNAI*. Springer, 2001.

Appendix: The proof of correctness & completeness

Some further notation

Let ψ be a K_m-formula. We denote by $\overline{\psi}$ the representation of $\neg\psi$ in the current formalism: in NNF, $\overline{\Diamond_r\psi} := \Box_r\overline{\psi}$, $\overline{\Box_r\psi} := \Diamond_r\overline{\psi}$, $\overline{\psi_1 \wedge \psi_2} := \overline{\psi_1} \vee \overline{\psi_2}$, $\overline{\psi_1 \vee \psi_2} := \overline{\psi_1} \wedge \overline{\psi_2}$, $\overline{A_i} := \neg A_i$, $\overline{\neg A_i} := A_i$; in BNF, $\overline{\neg\Box_r\psi} := \Box_r\psi$, $\overline{\Box_r\psi} := \neg\Box_r\psi$, $\overline{\psi_1 \wedge \psi_2} := \overline{\psi_1} \vee \overline{\psi_2}$, $\overline{\psi_1 \vee \psi_2} := \overline{\psi_1} \wedge \overline{\psi_2}$, $\overline{A_i} := \neg A_i$, $\overline{\neg A_i} := A_i$.

We represent a truth assignment μ as a set of literals, with the intended meaning that a positive literal A_i (resp. negative literal $\neg A_i$) in μ means that A_i is assigned to true (resp. false). We say that μ *assigns* a literal l if it assigns a truth value to the atom of l. We say that a literal l occurs in a boolean formula ϕ iff its atom occurs in ϕ.

Let $\mathcal{M}$ denote a Kripke model, and let σ (the label of) a generic state in $\mathcal{M}$. We denote by "1" is the root state of $\mathcal{M}$. By "$\langle\sigma : \psi\rangle \in \mathcal{M}$" we mean that $\sigma \in \mathcal{M}$ and $\mathcal{M}, \sigma \models \psi$. Thus, for every $\sigma \in \mathcal{M}$, either $\langle\sigma : \psi\rangle \in \mathcal{M}$ or $\langle\sigma : \overline{\psi}\rangle \in \mathcal{M}$.

For convenience, instead of (7) sometimes we use the equivalent definition:

$$Def(\sigma,\, \nu^r) := (L_{[\sigma,\, \nu^r]} \rightarrow \bigwedge_{\langle\sigma:\pi^{r,i}\rangle} (L_{[\sigma,\, \pi^{r,i}]} \rightarrow L_{[\sigma.i,\, \nu^r_0]})) \wedge \bigwedge_{\langle\sigma:\pi^{r,i}\rangle} Def(\sigma.i,\, \nu^r_0). \quad (9)$$

Notice that each $Def(\sigma,\, \psi)$ in (4), (5), (6), (9) is written in the general form

$$(L_{[\sigma,\, \psi]} \rightarrow \Phi_{\langle\sigma:\psi\rangle}) \wedge \bigwedge_{\langle\sigma':\psi'\rangle} Def(\sigma',\, \psi'). \quad (10)$$

We call *definition implication* for $Def(\sigma,\, \psi)$ the expressions "$(L_{[\sigma,\, \psi]} \rightarrow \Phi_{\langle\sigma:\psi\rangle})$".

The proof

Let φ be a K_m-formula. We prove the following theorem, which states the soundness and completeness of K_m2SAT.

Theorem 2. *A K_m-formula φ is K_m-satisfiable if and only if $K_m2SAT(\varphi)$ is satisfiable.*

Proof. It is a direct consequence of the following Lemmas 1 and 2. □.

Lemma 1. *Given a K_m-formula φ, if $K_m2SAT(\varphi)$ is satisfiable, then there exists a Kripke model $\mathcal{M}$ s.t. $\mathcal{M},1 \models \varphi$.*

Proof. Let μ be a total truth assignment satisfying $K_m2SAT(\varphi)$. We build from μ a Kripke model $\mathcal{M} = \langle \mathcal{U}, \mathcal{L}, \mathcal{R}_1, \ldots, \mathcal{R}_m \rangle$ as follows:

$$\mathcal{U} := \{\sigma \ : \ A_{[\sigma,\ \psi]} \textit{ occurs in } K_m2SAT(\varphi) \textit{ for some } \psi\} \tag{11}$$

$$\mathcal{L}(\sigma, A_i) := \begin{cases} True & if\ L_{[\sigma,\ A_i]} \in \mu \\ False & if\ \neg L_{[\sigma,\ A_i]} \in \mu \end{cases} \tag{12}$$

$$\mathcal{R}_x := \{\langle \sigma, \sigma.i \rangle \ : \ L_{[\sigma,\ \pi^{r,i}]} \in \mu\}. \tag{13}$$

We show by induction on the structure of φ that, for every $\langle \sigma : \psi \rangle$ s.t. $L_{[\sigma,\ \psi]}$ occurs on $K_m2SAT(\varphi)$,

$$\langle \sigma : \psi \rangle \in \mathcal{M} \ \text{ if } \ L_{[\sigma,\ \psi]} \in \mu. \tag{14}$$

Base.

$\psi = A_i$ or $\psi = \neg A_i$. Then (14) follows trivially from (12).

Step.

$\psi = \alpha$. Let $L_{[\sigma,\ \alpha]} \in \mu$.
  As μ satisfies (4), $L_{[\sigma,\ \alpha_i]} \in \mu$ for every $i \in \{1,2\}$.
  By inductive hypothesis, $\langle \sigma : \alpha_i \rangle \in \mathcal{M}$ for every $i \in \{1,2\}$.
  Then, by definition, $\langle \sigma : \alpha \rangle \in \mathcal{M}$.
  Thus, $\langle \sigma : \alpha \rangle \in \mathcal{M}$ if $L_{[\sigma,\ \alpha]} \in \mu$.

$\psi = \beta$. Let $L_{[\sigma,\ \beta]} \in \mu$.
  As μ satisfies (5), $L_{[\sigma,\ \beta_i]} \in \mu$ for some $i \in \{1,2\}$.
  By inductive hypothesis, $\langle \sigma : \beta_i \rangle \in \mathcal{M}$ for some $i \in \{1,2\}$.
  Then, by definition, $\langle \sigma : \beta \rangle \in \mathcal{M}$.
  Thus, $\langle \sigma : \beta \rangle \in \mathcal{M}$ if $L_{[\sigma,\ \beta]} \in \mu$.

$\psi = \pi^{r,j}$. Let $L_{[\sigma,\ \pi^{r,j}]} \in \mu$.
  As μ satisfies (6), $L_{[\sigma.j,\ \pi^{r,j}]} \in \mu$.
  By inductive hypothesis, $\langle \sigma.j : \pi_0^{r,j} \rangle \in \mathcal{M}$.
  Then, by definition and by (13), $\langle \sigma : \pi^{r,j} \rangle \in \mathcal{M}$.
  Thus, $\langle \sigma : \pi^{r,j} \rangle \in \mathcal{M}$ if $L_{[\sigma,\ \pi^{r,j}]} \in \mu$.

$\psi = \nu^r$. Let $L_{[\sigma,\ \nu^r]} \in \mu$.
  As μ satisfies (7), for every $\langle \sigma : \pi^{r,i} \rangle$ s.t. $L_{[\sigma,\ \pi^{r,i}]} \in \mu$, we have that $L_{[\sigma.i,\ \nu_0^r]} \in \mu$.
  By inductive hypothesis, we have that $\langle \sigma : \pi^{r,i} \rangle \in \mathcal{M}$ and $\langle \sigma.i : \nu_0^r \rangle \in \mathcal{M}$.
  Then, by definition and by (13), $\langle \sigma : \nu^r \rangle \in \mathcal{M}$.
  Thus, $\langle \sigma : \nu^r \rangle \in \mathcal{M}$ if $L_{[\sigma,\ \nu^r]} \in \mu$.

If $\mu \models K_m2SAT(\varphi)$, then $A_{[1,\ \varphi]} \in \mu$. Thus, by (14), $\langle 1 : \varphi \rangle \in \mathcal{M}$, i.e., $\mathcal{M},1 \models \varphi$. □

Lemma 2. *Given a K_m-formula φ, if there exists a Kripke model $\mathcal{M}$ s.t. $\mathcal{M},1 \models \varphi$, then $K_m2SAT(\varphi)$ is satisfiable.*

Proof. Let $\mathcal{M}$ be a Kripke model s.t. $\mathcal{M},1 \models \varphi$. We build from $\mathcal{M}$ a truth assignment μ for $K_m2SAT(\varphi)$ recursively as follows: [15]

$$\mu := \mu_{\mathcal{M}} \cup \mu_{\overline{\mathcal{M}}} \tag{15}$$

$$\begin{aligned}\mu_{\mathcal{M}} :=\ & \{L_{[\sigma,\,\psi]} \in K_m2SAT(\varphi)\ :\ \langle\sigma:\psi\rangle \in \mathcal{M}\} \\ & \cup\ \{\neg L_{[\sigma,\,\psi]} \in K_m2SAT(\varphi)\ :\ \langle\sigma:\overline{\psi}\rangle \in \mathcal{M}\}\end{aligned} \tag{16}$$

$$\mu_{\overline{\mathcal{M}}} := \mu_{\pi\nu} \cup \mu_{\alpha\beta} \tag{17}$$

$$\begin{aligned}\mu_{\pi\nu} :=\ & \{\neg L_{[\sigma,\,\pi^{r,i}]} \in K_m2SAT(\varphi)\ :\ \sigma \notin \mathcal{M}\} \\ & \cup\ \{L_{[\sigma,\,\nu^r]} \in K_m2SAT(\varphi)\ :\ \sigma \notin \mathcal{M}\}\end{aligned} \tag{18}$$

$$\begin{aligned}\mu_{\alpha\beta} :=\ & \{L_{[\sigma,\,\alpha]} \in K_m2SAT(\varphi)\ :\ \sigma \notin \mathcal{M} \textit{ and } L_{[\sigma,\,\alpha_i]} \in \mu_{\overline{\mathcal{M}}} \textit{ for every } i \in \{1,2\}\} \\ & \cup\ \{\neg L_{[\sigma,\,\alpha]} \in K_m2SAT(\varphi)\ :\ \sigma \notin \mathcal{M} \textit{ and } \neg L_{[\sigma,\,\alpha_i]} \in \mu_{\overline{\mathcal{M}}} \textit{ for some } i \in \{1,2\}\} \\ & \cup\ \{L_{[\sigma,\,\beta]} \in K_m2SAT(\varphi)\ :\ \sigma \notin \mathcal{M} \textit{ and } L_{[\sigma,\,\beta_i]} \in \mu_{\overline{\mathcal{M}}} \textit{ for some } i \in \{1,2\}\} \\ & \cup\ \{\neg L_{[\sigma,\,\beta]} \in K_m2SAT(\varphi)\ :\ \sigma \notin \mathcal{M} \textit{ and } \neg L_{[\sigma,\,\beta_i]} \in \mu_{\overline{\mathcal{M}}} \textit{ for every } i \in \{1,2\}\}.\end{aligned} \tag{19}$$

By construction, for every $L_{[\sigma,\,\psi]}$ in $K_m2SAT(\varphi)$, μ assigns $L_{[\sigma,\,\psi]}$ to true iff it assigns $L_{[\sigma,\,\overline{\psi}]}$ to false and vice versa, so that μ is a consistent truth assignment.

First, we show that $\mu_{\mathcal{M}}$ satisfies the definition implications of all $Def(\sigma,\,\psi)$'s and $Def(\sigma,\,\overline{\psi})$' s.t. $\sigma \in \mathcal{M}$. Let $\sigma \in \mathcal{M}$. We distinguish four cases.

$\psi = \alpha$. Thus $\overline{\psi} = \beta$ s.t. $\beta_1 = \overline{\alpha_1}$ and $\beta_2 = \overline{\alpha_2}$.

- If $\langle\sigma:\alpha\rangle \in \mathcal{M}$ (and hence $\langle\sigma:\beta\rangle \notin \mathcal{M}$), then for both i's $\langle\sigma:\alpha_i\rangle \in \mathcal{M}$ and $\langle\sigma:\beta_i\rangle \notin \mathcal{M}$. Thus, by (16), $\{L_{[\sigma,\,\alpha_1]}, L_{[\sigma,\,\alpha_2]}, \neg L_{[\sigma,\,\beta]}\} \subseteq \mu_{\mathcal{M}}$, so that $\mu_{\mathcal{M}}$ satisfies the definition implications of both $Def(\sigma,\,\alpha)$ and $Def(\sigma,\,\beta)$.
- If $\langle\sigma:\alpha\rangle \notin \mathcal{M}$ (and hence $\langle\sigma:\beta\rangle \in \mathcal{M}$), then for some i $\langle\sigma:\alpha_i\rangle \notin \mathcal{M}$ and $\langle\sigma:\beta_i\rangle \in \mathcal{M}$. Thus, by (16), $\{\neg L_{[\sigma,\,\alpha]}, L_{[\sigma,\,\beta_i]}\} \subseteq \mu_{\mathcal{M}}$, so that $\mu_{\mathcal{M}}$ satisfies the definition implications of both $Def(\sigma,\,\alpha)$ and $Def(\sigma,\,\beta)$.

$\psi = \beta$. Like in the previous case, inverting ψ and $\overline{\psi}$.

$\psi = \pi^{r,j}$. Thus $\overline{\psi} = \nu^r$ s.t. $\nu^r_0 = \overline{\pi^{r,j}_0}$.

- If $\langle\sigma:\pi^{r,j}\rangle \in \mathcal{M}$ (and hence $\langle\sigma:\nu^r\rangle \notin \mathcal{M}$), then $\langle\sigma.j:\pi^{r,j}_0\rangle \in \mathcal{M}$. Thus, by (16), $\{L_{[\sigma.j,\,\pi^{r,j}_0]}, \neg L_{[\sigma,\,\nu^r]}\} \subseteq \mu_{\mathcal{M}}$, so that $\mu_{\mathcal{M}}$ satisfies the definition implications of both $Def(\sigma,\,\pi^{r,j})$ and $Def(\sigma,\,\nu^r)$.
- If $\langle\sigma:\pi^{r,j}\rangle \notin \mathcal{M}$ (and hence $\langle\sigma:\nu^r\rangle \in \mathcal{M}$), then by (16) $\neg L_{[\sigma,\,\pi^{r,j}]} \in \mu_{\mathcal{M}}$, so that $\mu_{\mathcal{M}}$ satisfies the definition implications of $Def(\sigma,\,\pi^{r,j})$.

[15] We assume that $\mu_{\mathcal{M}}$, $\mu_{\pi\nu}$ and $\mu_{\alpha\beta}$ are generated *in order*, so that $\mu_{\alpha\beta}$ is generated recursively starting from $\mu_{\pi\nu}$. Intuitively, $\mu_{\mathcal{M}}$ assigns the literals $L_{[\sigma,\,\psi]}$ s.t. $\sigma \in \mathcal{M}$ in such a way to mimic $\mathcal{M}$; $\mu_{\overline{\mathcal{M}}}$ assigns the other literals in such a way to mimic the fact that no state outside those in $\mathcal{M}$ is generated (i.e., all $L_{[\sigma,\,\pi]}$'s are assigned false and the $L_{[\sigma,\,\nu]}$'s, $L_{[\sigma,\,\alpha]}$'s, $L_{[\sigma,\,\beta]}$'s are assigned consequently).

As far as $Def(\sigma,\ \nu^r)$ is concerned, we partition the clauses in (7):

$$((L_{[\sigma,\ \nu^r]} \wedge L_{[\sigma,\ \pi^{r,i}]}) \rightarrow L_{[\sigma.i,\ \nu_0^r]}) \tag{20}$$

into two subsets. The first is the set of clauses (20) for which $\langle \sigma : \pi^{r,i} \rangle \in \mathcal{M}$. As $\langle \sigma : \nu^r \rangle \in \mathcal{M}$, we have that $\langle \sigma.i : \nu_0^r \rangle \in \mathcal{M}$. Thus, by (16), $L_{[\sigma.i,\ \nu_0^r]} \in \mu_{\mathcal{M}}$, so that $\mu_{\mathcal{M}}$ satisfies (20). The second is the set of clauses (20) for which $\langle \sigma : \pi^{r,i} \rangle \notin \mathcal{M}$. By (16) we have that $\neg L_{[\sigma,\ \pi^{r,i}]} \in \mu_{\mathcal{M}}$, so that $\mu_{\mathcal{M}}$ satisfies (20). Thus, $\mu_{\mathcal{M}}$ satisfies the definition implications also of $Def(\sigma,\ \nu^r)$.

$\psi = \nu^r$. Like in the previous case, inverting ψ and $\overline{\psi}$.

Notice that, if $\sigma \notin \mathcal{M}$, then $\sigma.i \notin \mathcal{M}$ for every i. Thus, for every $Def(\sigma,\ \psi)$ s.t. $\sigma \notin \mathcal{M}$, all atoms in the implication definition of $Def(\sigma,\ \psi)$ are not assigned by $\mu_{\mathcal{M}}$.

Second, we show by induction on the recursive structure of $\mu_{\overline{\mathcal{M}}}$ that $\mu_{\overline{\mathcal{M}}}$ satisfies the definition implications of all $Def(\sigma,\ \psi)$'s and $Def(\sigma,\ \overline{\psi})$'s s.t. $\sigma \notin \mathcal{M}$. Let $\sigma \notin \mathcal{M}$.

As a base step, by (18), $\mu_{\pi\nu}$ satisfies the definition implications of all $Def(\sigma,\ \pi^{r,i})$'s and $Def(\sigma,\ \nu^r)$'s because it assigns false to all $L_{[\sigma,\ \pi^{r,i}]}$'s.

As inductive step, we show on the inductive structure of $\mu_{\alpha\beta}$ that $\mu_{\alpha\beta}$ satisfies the definition implications of all $Def(\sigma,\ \alpha)$'s and $Def(\sigma,\ \beta)$'s. Let $\psi := \alpha$ and $\overline{\psi} = \beta$ s.t. $\beta_i = \overline{\alpha_i}$ (or vice versa). Then we have that:

- if both $L_{[\sigma,\ \alpha_i]}$'s are assigned true by $\mu_{\overline{\mathcal{M}}}$ (and hence both $L_{[\sigma,\ \beta_i]}$'s are assigned false), then by (19) $L_{[\sigma,\ \alpha]}$ is assigned true and $L_{[\sigma,\ \beta]}$ is assigned false by $\mu_{\alpha\beta}$, which satisfies the definition implications of both $Def(\sigma,\ \alpha)$ and $Def(\sigma,\ \beta)$;
- if one $L_{[\sigma,\ \alpha_i]}$ is assigned false by $\mu_{\overline{\mathcal{M}}}$ (and hence $L_{[\sigma,\ \beta_i]}$ is assigned true), then by (19) $L_{[\sigma,\ \alpha]}$ is assigned false and $L_{[\sigma,\ \beta]}$ is assigned true by $\mu_{\alpha\beta}$, which satisfies the definition implications of both $Def(\sigma,\ \alpha)$ and $Def(\sigma,\ \beta)$.

Thus $\mu_{\overline{\mathcal{M}}}$ satisfies the definition implications of all $Def(\sigma,\ \psi)$'s and $Def(\sigma,\ \overline{\psi})$'s s.t. $\sigma \notin \mathcal{M}$.

On the whole, $\mu \models Def(\sigma,\ \psi)$ for every $Def(\sigma,\ \psi)$. By construction, $\mu_{\mathcal{M}} \models A_{[1,\ \varphi]}$. Therefore $\mu \models K_m 2SAT(\varphi)$. □

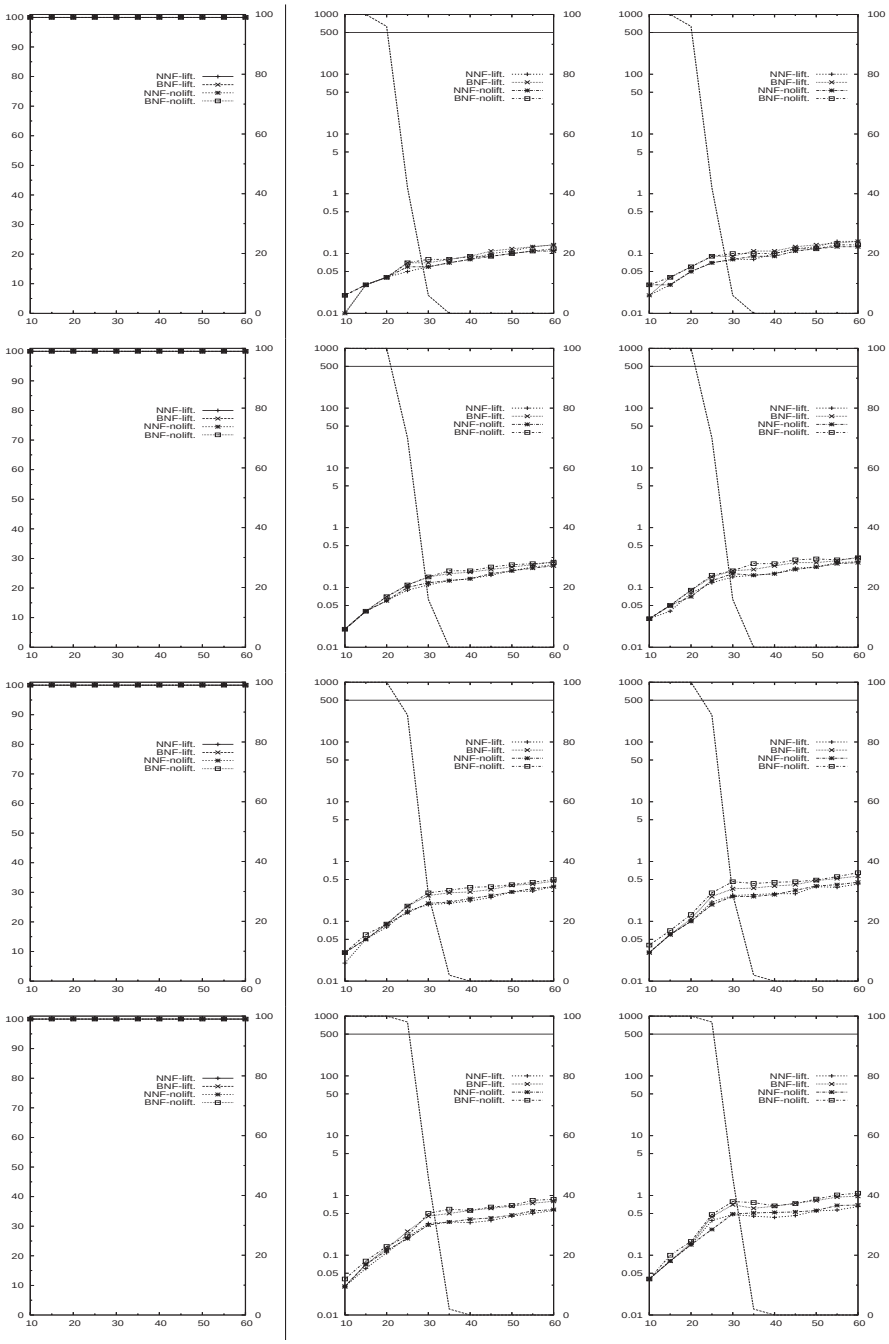

Fig. 1. Comparison among different variants of K_m2SAT +ZCHAFF on random problems, $d = 1$, $p = 0.5$, 100 samples/point. X axis: $\#clauses/N$. Y axis: 1st column: % of problems solved within the timeout; 2nd and 3rd columns: CPU time, 50th and 90th percentiles. 1st to 4th row: $N = 6, 7, 8, 9$. Background: % of satisfiable instances.

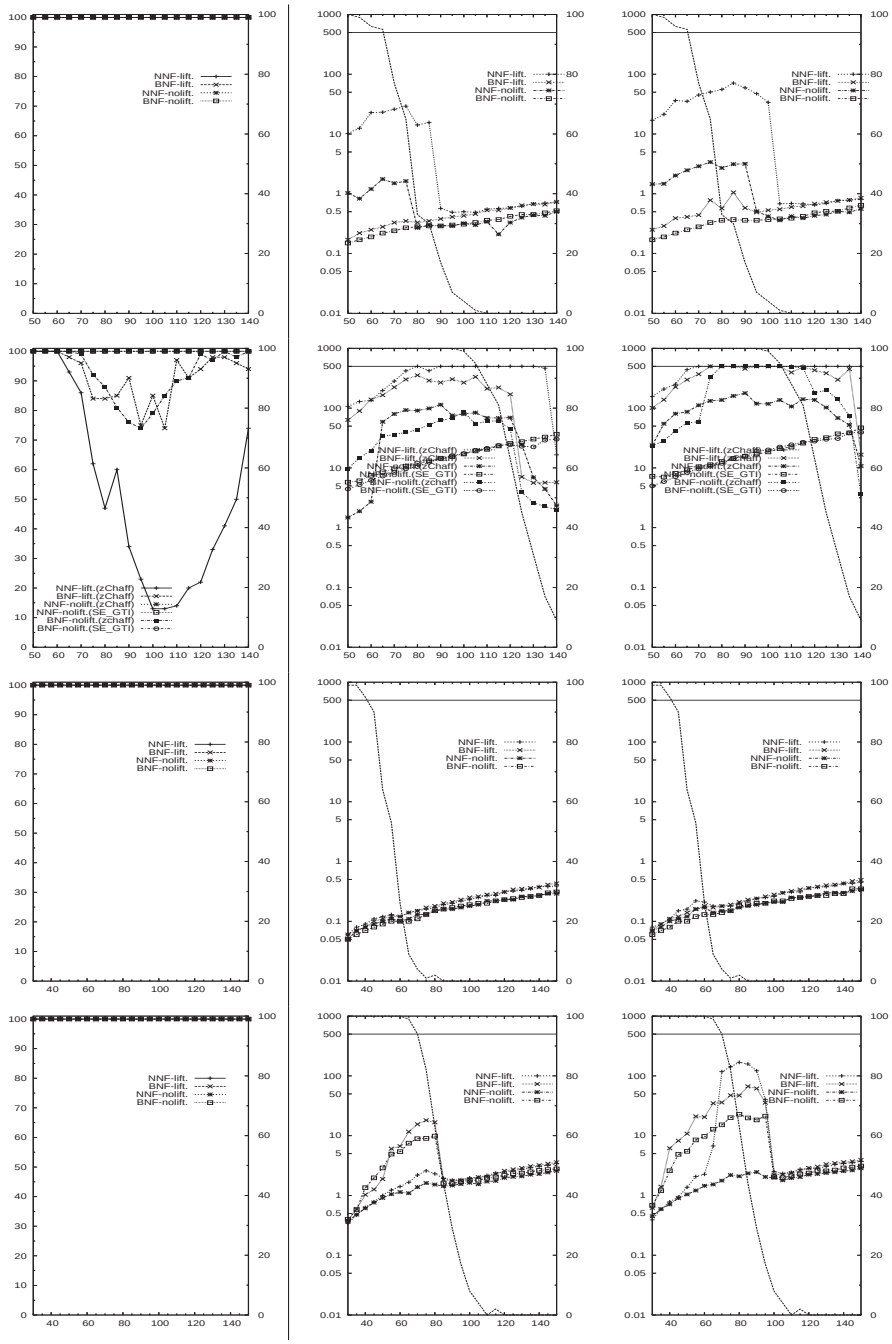

Fig. 2. Comparison among different variants of K_m2SAT +ZCHAFF on random problems, $d = 2$, 100 samples/point. X axis: $\#clauses/N$. Y axis: 1st column: % of problems solved within the timeout; 2nd and 3rd columns: CPU time, 50th and 90th percentiles. 1st and 2nd row: $p = 0.5$, $N = 3,4$; 3rd and 4th row: $p = 0.6$, $N = 3,4$. Background: % of satisfiable instances.

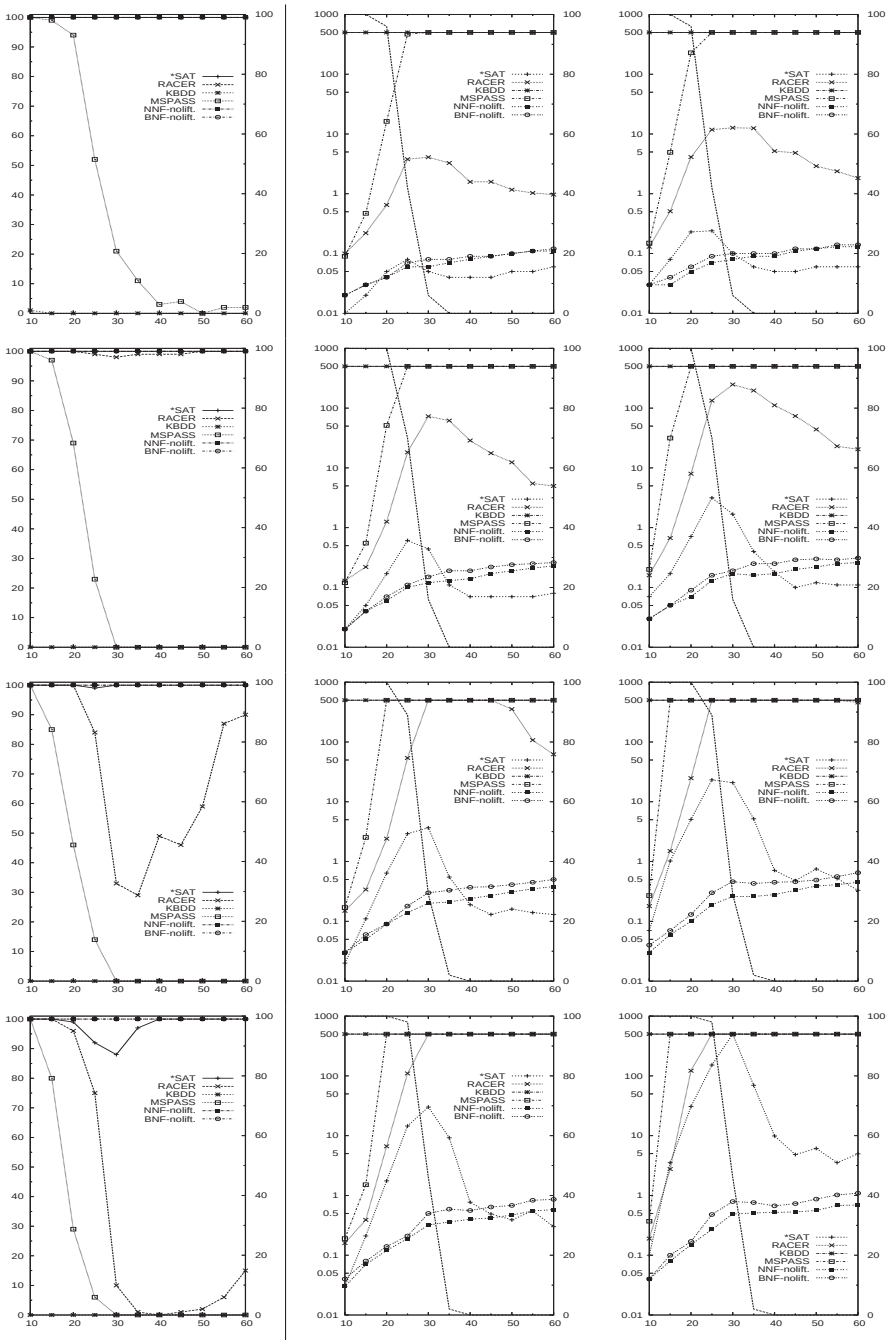

Fig. 3. Comparison against other approaches on random problems, $d = 1$, $p = 0.5$, 100 samples/point. X axis: $\#clauses/N$. Y axis: 1st column: % of problems solved within the timeout; 2nd and 3rd columns: CPU time, 50th and 90th percentiles. 1st to 4th row: $N = 6, 7, 8, 9$. Background: % of satisfiable instances.

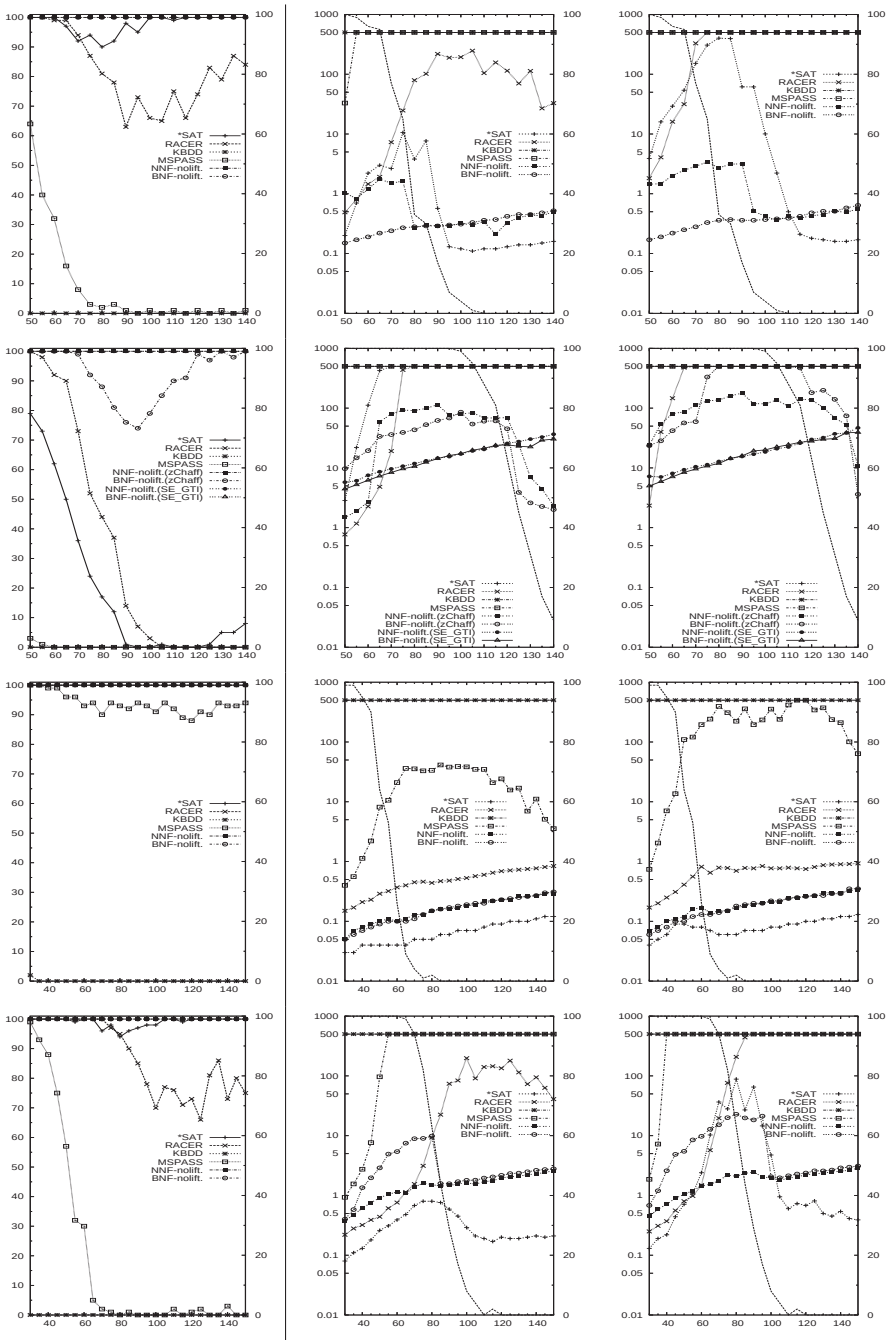

Fig. 4. Comparison against other approaches on random problems, $d = 2$, 100 samples/point. X axis: $\#clauses/N$. Y axis: 1st column: % of problems solved within the timeout; 2nd and 3rd columns: CPU time, 50th and 90th percentiles. 1st and 2nd row: $p = 0.5$, $N = 3,4$; 3rd and 4th row: $p = 0.6$, $N = 3,4$. Background: % of satisfiable instances.