Building decision procedures for modal logics from propositional decision procedures — the case study of modal K *

Fausto Giunchiglia[1] Roberto Sebastiani[2]

[1] IRST, 38050 Povo (TN), Italy. ph: ++39.461.314436
[2] DIST, University of Genoa, v. Causa 13, 16146 Genoa, Italy. ph: ++39.10.3532811
fausto@irst.itc.it rseba@mrg.dist.unige.it

Abstract. The goal of this paper is to propose a new technique for developing decision procedures for propositional modal logics. The basic idea is that propositional modal decision procedures should be developed on top of propositional decision procedures. As a case study, we describe an algorithm, based on an implementation of the Davis-Putnam-Longemann-Loveland procedure, which tests satisfiability in modal K. The algorithm is compared with a tableau based decision procedure. The experimental results show that our algorithm outperforms this system. The testing is performed following a newly developed methodology which, among other things, allows us to classify problems according to an easy to hard pattern.

1 Introduction

The goal of this paper is to describe a new technique for developing decision procedures for propositional modal logics. Our approach is based on two basic intuitions. The first is that modal reasoning can be implemented as an "appropriate composition" of reasoning inside multiple propositional theories (or models, if one thinks of satisfiability). [GS94] shows how this can be done for provability in the most common normal modal logics; [GSGF93] extends these results to various non normal modal logics. Similar ideas are implicit, even if never spelled out as such, in the tableaux for normal modal logics described in [Fit88, Mas94]. The second is that propositional reasoning can be performed very efficiently by using state-of-the-art propositional decision procedures, e.g., Davis-Putnam-Longemann-Loveland (DPLL from now on) [DLL62], OBDD [Bry92], Dynamic Backtracking [Gin93] decision procedures, or even partial decision procedures

* This work has benefited from many long discussions with Enrico Giunchiglia. Enrico played a crucial role in the development of Version 1 of the algorithm described in this paper. Bernhard Nebel and Enrico Franconi have given us access to the tableau implementation used in the tests described in this paper. Marco Roveri has given technical assistance in the testing phase. All the members of the Mechanized Reasoning Group in Genoa are thanked for putting up with many weeks of CPU-time background processes.

like GSAT [SLM92]. This allows us to exploit the huge amount of technology developed in this area, which is very advanced and well understood. Furthermore, the multiple calls to propositional decision procedures can be made very efficient by storing and propagating partial common computations.

In this paper we concentrate on the satisfiability problem for modal K (from now on, also called K-satisfiability), and test propositional satisfiability (not restricted to CNF) using an implementation of DPLL [1]. The reasons for this choice are manyfold. First, K is an interesting logic per se; for instance it is well known that K(m) (that is, K with m modalities) is a notational variant of the terminological logic $\mathcal{ALC}$ [SSS91, Sch91]. (This is actually the main motivation for this work.) Second, K is the smallest normal modal logic. The algorithm(s) described in this paper can be (more or less trivially) extended to the other normal modal logics, for instance along the lines of what described in [Fit88], [Mas94] or [GS94]. Finally, recent experimental results show that DPLL is one of the most efficient propositional decision procedures [BB92] (but see also [US94]).

This paper is structured as follows. In Section 2 we describe the algorithm. This presentation is done incrementally, first by giving the algorithm implementing the basic idea, and then by providing three enhancements, each improving on a specific aspect. Section 3 describes a methodology we have developed for generating test problems. This methodology extends a methodology developed for propositional satisfiability and, among other things, allows us to classify problems according to an easy to hard pattern. An extensive search conducted before starting this work indicated that this is the first time that such a methodology has been developed for modal logics, and in particular for K. Worse than this, it also highlighted the fact that very few test examples could be found in the literature. In Section 4 our algorithm is tested against a tableau based decision procedure. The experimental results show that our algorithm outperforms this procedure by orders of magnitude. Finally, section 5 provides some conclusive remarks and describes the directions for future work.

2 The Algorithm(s)

Let us explain the intuition underlying our algorithm in terms of Fitting's tableau for K, as described in [Fit88]. Let us consider only the $\Box$ operator. Fitting's tableau works by refutation; the idea is to prove that a well formed formula (wff from now on) is unsatisfiable. Its rules can be divided into two sets:

1. a set of rules complete for propositional logic. The idea is to manipulate signed wffs, where a signed wff is a wff prefixed by either T (true) or F (false), intuitively indicating that the wff is true or false. There is a set of rules which allows us to add wffs to branches, and to split branches. A branch is closed if it contains $T\psi$ and $F\psi$, for some wff ψ. (From now on we use greek letters, e.g., α, β, φ, ψ, to represent generic modal wffs, and capital

[1] [AG93] and [Seb94] show how decision procedures for CNF formulas like DPLL and GSAT can be modified to work for non-CNF formulas.

letters, e.g., A_1, A_2, to represent propositional atoms.) A tableau proof starts with the wff to be proved, prefixed by F; the wff is proved if the tableau is closed, i.e., if all the branches are closed.

2. a set of rules which defines the modal logic. This consists of (a singleton set containing) a rule of *destructive substitution* which allows us to substitute a set of (signed) wffs on a tableau branch with a new set of signed wffs. The idea is that all the wffs of form $T\Box\psi$ are substituted with $T\psi$, a chosen wff of the form $F\Box\varphi$ is substituted with $F\varphi$, and all the other wffs of the form $F\Box\varphi$ are forgotten. There is a catch that we may have to do as many destructive substitutions as there are wffs of the form $F\Box\varphi$ (the rule of destructive substitution must be iterated till we find a branch which closes).

The key observation is that the wffs of the form $\Box\psi$ are treated as atomic wffs by the rules which implement the propositional reasoning. There is no way to look into the argument of $\Box$. For instance $\Box(\psi_1 \wedge \psi_2)$ and $\Box(\psi_2 \wedge \psi_1)$ or, even, $\Box\psi$ and $\Box(\psi \wedge \psi)$ are treated as different wffs. The specific form of ψ in $\Box\psi$ matters only when the $\Box$ is stripped off by the rules which implement the modal reasoning (in Fitting's tableau, the rule of destructive substitution) [2].

2.1 Version 1: the basic algorithm

Let us call *atom* any wff which cannot be decomposed propositionally, and *modal atom* any atom of the form $\Box\psi$. Let φ be the modal wff to be proved satisfiable. Then the (key) observation made above in this section suggests the following two basic steps for testing K-satisfiability, the first implementing the propositional reasoning, the second implementing the modal reasoning:

1. Using a decision procedure for propositional satisfiability, assign a truth value to the atoms occurring in φ in a way to make φ evaluate to T. Let us call *truth assignment (for φ)* the resulting set μ of truth value assignments. We say that μ (propositionally) *satisfies* φ. Then μ is of the form

$$\mu = \{ \Box\alpha_1 = T, \ldots, \Box\alpha_n = T, \Box\beta_1 = F, \ldots, \Box\beta_m = F, \\ A_1 = T, \ldots, A_r = T, A_{r+1} = F, \ldots, A_s = F\}$$

Notationally, from now on we write μ as

$$\mu = \bigwedge_i \Box\alpha_i \;\wedge \bigwedge_j \neg\Box\beta_j \;\wedge \gamma \tag{1}$$

where $\gamma = \bigwedge_{k=1}^r A_k \;\wedge \bigwedge_{h=r+1}^s \neg A_h$ is a conjunction of propositional literals. Furthermore we use the greek letters μ, η to represent truth assignments.

2. Prove that the input wff φ is K-satisfiable by finding (among all the possible truth assignments) a truth assignment μ such that, for all β_j's, the wff $\bigwedge_i \alpha_i \wedge \neg\beta_j$ is K-satisfiable. If no truth assignment is found which is K-satisfiable, then φ is not K-satisfiable (K-unsatisfiable).

[2] This observation was first made by looking at the systems defined in [GS94]. These systems allow for the use of multiple distinct propositional theories, where the role of $\Box\psi$ is played by the atomic wff $T(\text{“}\psi\text{”})$, with T a unary predicate.

The two steps recurse until we get to a truth assignment with no modal atoms.

Example 1. Consider the following modal wff φ:

$$\begin{aligned}
\varphi = \{ & \underline{\neg\Box(\neg A_3 \vee \neg A_1 \vee A_2)} \vee A_1 \vee A_5\} \wedge \\
\{ & \underline{\neg A_2} \vee \neg A_5 \vee \Box(\neg A_2 \vee \neg A_4 \vee \neg A_3)\} \wedge \\
\{ & \underline{A_1} \vee \neg\Box(\neg A_4 \vee A_5 \vee A_2) \vee A_2\} \wedge \\
\{ & \neg\Box(A_4 \vee \neg A_3 \vee A_1) \vee \underline{\neg\Box(A_4 \vee \neg A_2 \vee A_3)} \vee \neg A_5\} \wedge \\
\{ & \underline{\neg A_3} \vee A_1 \vee \neg\Box(\neg A_4 \vee \neg A_2 \vee A_2)\} \wedge \\
\{ & \underline{\Box(\neg A_5 \vee A_4 \vee A_3)} \vee \Box(\neg A_1 \vee A_4 \vee A_3) \vee \neg A_1\} \wedge \\
\{ & A_1 \vee \underline{\Box(\neg A_2 \vee A_1 \vee A_4)} \vee A_2\}
\end{aligned}$$

Consider the following truth assignment μ_1, which sets to T the literals in φ which are underlined:

$$\begin{aligned}
\mu_1 = {} & \Box(\neg A_5 \vee A_4 \vee A_3) \wedge \quad \Box(\neg A_2 \vee A_1 \vee A_4) \wedge & & [\textstyle\bigwedge_i \Box\alpha_i] \\
& \neg\Box(\neg A_3 \vee \neg A_1 \vee A_2) \wedge \neg\Box(A_4 \vee \neg A_2 \vee A_3) \wedge & & [\textstyle\bigwedge_j \neg\Box\beta_j] \\
& A_1 \wedge \neg A_2 \wedge \neg A_3. & & [\gamma]
\end{aligned}$$

μ_1 satisfies φ. Then μ_1 is K-satisfiable iff each of the wffs

$$\begin{aligned}
\textstyle\bigwedge_i \alpha_i \wedge \neg\beta_1 &= (\neg A_5 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge A_3 \wedge A_1 \wedge \neg A_2, \\
\textstyle\bigwedge_i \alpha_i \wedge \neg\beta_2 &= (\neg A_5 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \neg A_4 \wedge A_2 \wedge \neg A_3
\end{aligned}$$

is K-consistent. As they both are, then φ is K-consistent.

The basic version of our algorithm is implemented by the function KSAT in Figure 1. KSAT takes in input a modal propositional wff φ and returns a truth value asserting whether φ is K-satisfiable or not. KSAT invokes directly KSAT_{WFF}, passing as arguments φ and the truth value T (i.e., by (1), the empty truth assignment). KSAT_{WFF} tries to build a K-satisfiable truth assignment μ satisfying φ. This is done recursively, according to the following steps:

- (base) If $\varphi = T$, then μ satisfies φ. Thus, if μ is K-satisfiable, then φ is K-satisfiable. Therefore KSAT_{WFF} invokes $\text{KSAT}_{ASSGN}(\mu)$. KSAT_{ASSGN} returns a truth value asserting whether μ is K-satisfiable or not.
- (backtrack) If $\varphi = F$, then μ cannot be a truth assignment for φ. Therefore KSAT_{WFF} returns $False$.
- (unit) If a literal l occurs in φ as a unit clause, then l must be assigned T [3]. To obtain this, KSAT_{WFF} is invoked recursively with arguments the wff returned by *assign(l, φ)* and the assignment obtained by adding l to μ. *assign(l, φ)* substitutes every occurrence of l in φ with T and evaluates the result.
- (split) If none of the above situations occurs, then *choose-literal(φ)* returns an unassigned literal l according to some heuristic criterion. Then KSAT_{WFF} is first invoked recursively with arguments *assign(l, φ)* and $\mu \wedge l$. If the result is negative, then KSAT_{WFF} is invoked with arguments *assign($\neg l, \varphi$)* and $\mu \wedge \neg l$.

[3] A notion of unit clause for non-CNF propositional wffs is given in [AG93].

```
function KSAT(φ)
    return KSAT_WFF(φ, T);

function KSAT_WFF(φ, μ)
    if φ = T                                              /* base      */
        then return KSAT_ASSGN(μ);
    if φ = F                                              /* backtrack */
        then return False;
    if {a unit clause (l) occurs in φ}                    /* unit      */
        then return KSAT_WFF(assign(l, φ), μ ∧ l);
    l := choose-literal(φ);                               /* split     */
    return  KSAT_WFF(assign(l, φ), μ ∧ l)  or
            KSAT_WFF(assign(¬l, φ), μ ∧ ¬l);

function KSAT_ASSGN(⋀_i □α_i ∧ ⋀_j ¬□β_j ∧ γ)
    for any conjunct "¬□β_j" do
        if not KSAT(⋀_i α_i ∧ ¬β_j)
            then return False;
    return True;
```

Fig. 1. The basic version of KSAT.

The implementation of KSAT_{ASSGN} is a straightforward consequence of what written above in this section. Given a truth assignment $\mu = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$, $\text{KSAT}_{ASSGN}(\mu)$ invokes $\text{KSAT}(\bigwedge_i \alpha_i \wedge \neg\beta_j)$ for any $\Box\beta_j$ occurring in μ. This is repeated until either KSAT returns a negative value ($\text{KSAT}_{ASSGN}(\mu)$ returns $False$), or no more $\Box\beta_j$'s are available ($\text{KSAT}_{ASSGN}(\mu)$ returns $True$).

It is straightforward to see, for instance by comparing it with Fitting's tableau, that the algorithm discussed in Figure 1 is complete and correct for modal K. Furthermore, it is important to notice that KSAT_{WFF} is a variant of DPLL. Unlike DPLL, whenever an assignment μ has been found, KSAT_{WFF}, instead of returning $True$, invokes $\text{KSAT}_{ASSGN}(\mu)$. Essentially, DPLL is used to generate truth assignments, whose K-satisfiability is recursively checked by KSAT_{ASSGN}.

2.2 Version 2: ordering modal atoms

One of the main causes of inefficiency of the algorithm described so far is the number of truth assignments found by KSAT_{WFF}, which may be rather large. This is a direct consequence of the large number of distinct modal atoms which can occur inside a modal wff. Generally speaking, a solution which allows for a drastic reduction of distinct modal atoms could be to treat logically equivalent modal atoms as the same atom. Unfortunately, this would have unacceptable computational costs. Nevertheless, some low-cost preprocessing can be performed which collapses together trivially equivalent modal atoms. For instance, all modal atoms can be (internally) *ordered*, according to some order on sub-wffs.

(The specific ordering is irrelevant as long as there is one.) This avoids assigning different truth values to permutations of the same sub-wffs.

Example 2. Consider the modal atoms occurring in the wff φ in Example 1 (e.g., the atom $\Box(\neg A_3 \vee \neg A_1 \vee A_2)$). For any atom there are up to $3! = 6$ equivalent permutations.

2.3 Version 3: factorizing $\bigwedge_i \alpha_i$

In the schema of Figure 1, KSAT_{ASSGN} invokes repeatedly KSAT passing as arguments wffs of the form $\wedge_i\alpha_i \wedge \neg\beta_1, \ \ldots, \ \wedge_i\alpha_i \wedge \neg\beta_m$. All these wffs have the conjuncts $\wedge_i\alpha_i$ in common. At the j-th call, KSAT searches for a K-satisfiable truth assignment η_j satisfying $\wedge_i\alpha_i \wedge \neg\beta_j$. This is done from scratch, i.e., without trying to reuse any of the previously computed assignments $\eta_1 \ldots \eta_{j-1}$, or their restrictions to $\wedge_i\alpha_i$. The idea underlying Version 3 is to "factorize" the search of the truth assignments satisfying $\wedge_i\alpha_i$. Given $\alpha = \wedge_i\alpha_i$ and $B = \{\beta_1 \ldots \beta_m\}$, a propositional algorithm is used to find a sequence of truth assignments $\eta_1\ \eta_2 \ldots$ satisfying α. At the k-th truth assignment η_k, all the wffs β_j's "compatible" with η_k (i.e., such that $\eta_k \wedge \neg\beta_j$ is K-satisfiable) are discharged from B. This is iterated till B is empty (μ is K-satisfiable) or no more assignment η_k can be found (μ is not K-satisfiable).

In Figure 2 we present a revised version KSAT_{ASSGN}. As before, KSAT_{ASSGN} takes in input a truth assignment $\mu = \bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma$ and returns a truth value asserting whether μ is K-satisfiable or not. If no conjunct of the form $\neg\Box\beta_j$ occurs in μ, then μ is K-satisfiable, and thus KSAT_{ASSGN} returns $True$. Otherwise KSAT_{ASSGN} invokes the function *Uncompatible-Subset*, passing as arguments the wff $\bigwedge_i \alpha_i$, an empty assignment, and the (always non-empty) wff set $\{\beta_1 \ldots \beta_m\}$. *Uncompatible-Subset($\alpha, T, B$)* returns the set of the wffs β_j's in B which are not compatible with any truth assignment which satisfies α. KSAT_{ASSGN} returns $True$ if and only if this set is empty. Similarly to KSAT_{WFF}, *Uncompatible-Subset* tries to build truth assignments η's satisfying α. Whenever it finds one, all the wffs β_j's which are compatible with this assignment are discharged from B. Again, this is done recursively according to the following steps:

- (base) If $\alpha = T$, then KSAT is invoked on all wffs in the form $\eta \wedge \neg\beta_j$, for all β_j's in B. The set of the wffs β_j's in B which are not compatible with η (i.e., $\text{KSAT}(\eta \wedge \neg\beta_j)$ returns $False$) is then returned.
- (backtrack) If $\alpha = F$, then η does not satisfy α. Therefore *Uncompatible-Subset* returns the whole set B.
- (unit) If a literal l occurs in α as a unit clause (or equivalent form for non-CNF wffs) then l is added to η and *Uncompatible-Subset* is invoked recursively with *assign(l, α)*, $\eta \wedge l$ and B.
- (split) If none of the above situations occur, then *choose-literal(α)* returns an unassigned literal l. Then *Uncompatible-Subset* is first invoked with *assign(l, α)*, $\eta \wedge l$ and B, and the set returned is stored into B'. If B' is empty, then no

function $KSAT_{ASSGN}(\bigwedge_i \Box\alpha_i \wedge \bigwedge_j \neg\Box\beta_j \wedge \gamma)$
  if {there is no conjunct "$\neg\Box\beta_j$"}
    then return *True*;
  return *Is-Empty(Uncompatible-Subset(*$\bigwedge_i \alpha_i, T, \{\beta_1 \ldots \beta_m\}$*))*;

function *Uncompatible-Subset*(α, η, B)
  if $\alpha = T$ /* base */
    then return $\{\beta_j \in B \mid \mathrm{KSAT}(\eta \wedge \neg\beta_j) = False\}$;
  if $\alpha = F$ /* backtrack */
    then return B;
  if {a unit clause (l) is in α} /* unit */
    then return *Uncompatible-Subset(assign*$(l, \alpha), \eta \wedge l, B$*)*;
  l := *choose-literal*(α); /* split */
  B' := *Uncompatible-Subset(assign*$(l, \alpha), \eta \wedge l, B$*)* ;
  if *Is-Empty*(B')
    then return B';
    else return *Uncompatible-Subset(assign*$(\neg l, \alpha), \eta \wedge \neg l, B'$*)*;

Fig. 2. KSAT Version 3: a new schema for KSAT_{ASSGN}.

more uncompatible wffs are available, and thus B' is returned. Otherwise, *Uncompatible-Subset* is invoked with *assign*$(\neg l, \alpha)$, $\eta \wedge \neg l$ and B'.

Example 3. Consider φ and μ_1 as in Example 1. We have

$$\begin{aligned} \alpha &= (\underline{\neg A_5} \vee A_4 \vee A_3) \ \wedge (\underline{\neg A_2} \vee \underline{\underline{A_1}} \vee A_4) \\ \neg\beta_1 &= A_3 \wedge A_1 \wedge \neg A_2 \\ \neg\beta_2 &= \neg A_4 \wedge A_2 \wedge \neg A_3. \end{aligned}$$

Suppose *Uncompatible-Subset* selects in sequence the literals $\neg A_5$ and $\neg A_2$, finding thus the assignment $\eta_1 = \neg A_5 \wedge \neg A_2$ which satisfies α. Then $\eta_1 \wedge \neg\beta_1$ is satisfiable but $\eta_1 \wedge \neg\beta_2$ is not. For the "base" step, *Uncompatible-Subset* returns $B' = \{\beta_2\}$. Then *Uncompatible-Subset* splits on the literal $\neg A_2$, finding $\eta_2 = \neg A_5 \wedge A_2 \wedge A_1$. As $\eta_2 \wedge \neg\beta_2$ is satisfiable, *Uncompatible-Subset* returns the empty set. Therefore μ_1 is K-satisfiable.

Again, the reader may recognize in *Uncompatible-Subset* a variant of DPLL. The main difference is that *Uncompatible-Subset* returns a set of wffs instead of a truth value. As in KSAT_{WFF}, the base step is modified to use DPLL for generating truth assignments. Notice that the split step is asymmetric, as the second call is invoked with a smaller set B'.

2.4 Version 4: checking partial assignments

Despite the improvement brought by internally ordering modal atoms, the number of truth assignments found by KSAT_{WFF} may still be too large. Version 4 starts from the empirical observation that most assignments found by KSAT_{WFF}

are "trivially" K-unsatisfiable, that is, they will remain K-unsatisfiable even after removing some (or many) of their conjuncts. If a partial assignment μ' is K-unsatisfiable, then all its extensions are K-unsatisfiable. If the unsatisfiability of μ' is detected on time, then this prevents checking the K-satisfiability of all the up to $2^{M-|\mu'|}$ truth assignments which extend μ', being M the number of distinct (modal) atoms of φ and $|\mu'|$ the number of conjuncts in μ'.

This suggests the introduction of an intermediate K-satisfiability test on partial assignments just before the split. (Notice there is no need to introduce similar tests before unit propagation.) This can be done by introducing the three lines below in the function KSAT_{WFF} of Figure 1, between the "split" and the "unit":

```
if Likely-Unsatisfiable(μ)                    /* partial assigns check */
    if not KSAT_ASSGN(μ)
        then return False;
```

Let us temporarily ignore the test performed by *Likely-Unsatisfiable*. KSAT_{ASSGN} in invoked on the current partial assignment μ. If KSAT_{ASSGN} returns $False$, then all possible extensions of μ are unsatisfiable, and therefore KSAT_{WFF} returns $False$.

Example 4. Consider φ as in Example 1. Suppose that, after three recursive calls, KSAT_{WFF} builds the partial assignment

$$\mu_2' = \Box(\neg A_1 \vee A_4 \vee A_3) \wedge \Box(\neg A_2 \vee A_1 \vee A_4) \wedge \neg\Box(A_4 \vee \neg A_2 \vee A_3)$$

(rows 6, 7 and 4 of φ). If it is invoked on μ_2', KSAT_{ASSGN} will check the K-satisfiability of the single wff

$$\alpha \wedge \neg\beta_1 = (\neg A_1 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \neg A_4 \wedge A_2 \wedge \neg A_3$$

which is unsatisfiable. Therefore there will be no more need to select further literals, and KSAT_{WFF} will backtrack.

It may be argued that the introduction of an intermediate consistency check before *every* split could negatively affect the global worst-case performance. (In fact, in a binary tree the number of splitting internal nodes equals the number of leaves minus one.) In the hypothetical case in which no intermediate test caused backtracking, the number of KSAT_{ASSGN} calls per KSAT call could double, and the global number of KSAT_{ASSGN} calls might increase of up to a 2^{depth} factor, being $depth$ the modal depth of φ (where, by $depth$ of a modal wff φ, we mean the maximum number of nested "$\Box$" in φ).

To avoid this, it is worth introducing an heuristic function *Likely-Unsatisfiable*. The idea is that *Likely-Unsatisfiable* estimates μ being K-consistent according to parameter values like, e.g., the number of conjuncts and the number of propositional variables in μ. For instance, a simple heuristic could be to perform an intermediate check whenever the last literal added to μ is not propositional. More sophisticated heuristics could use previously-tabulated satisfiability transition diagrams like that reported in Figure 5.

3 The test method

In propositional and first order theorem proving there exists a wide bibliography on both problem sets (see, e.g., [SS95]) and test-generating methods (see, e.g., [BB92]). The solution we have adopted is to take a standard test-generation methodology for propositional CNF satisfiability (SAT) and extend it to modal K. The methodology we have considered is the *fixed clause-length model* (see, e.g., [MSL92, BB92]) with clause length $K = 3$ (*3-clause-length* from now on), briefly described below. Given a number N of propositional variables, for increasing values of the clause number L, sufficiently many (100, 500, 1000,...) random 3CNF wffs are generated and given in input to the procedure under test. After the computation, a statistical analysis on the satisfiability results and CPU times required (or equivalent data) is performed. The resulting statistical values, like satisfiability percentages and mean/median CPU times, are plotted against the L/N ratio. The process can be repeated for different numbers of propositional variables. Random 3CNF wffs are generated as follows: "for given L and N, an instance of a random 3SAT is produced by randomly generating L clauses of length 3. Each clause is produced by randomly choosing a set of 3 propositional variables from a set of N, and negating each with probability 0.5." (Quote from [MSL92]).

This method has three main features. First, there is a satisfiability-preserving way of converting all propositional wffs into 3CNF. Second, the number of parameters to handle reduces to 2. Finally, the parameters L and N allow for a coarse "tuning" of both the probability of satisfiability and the hardness of random 3CNF wffs. In fact, L [N] monotonically increases [decreases] the level of constraintness. Thus, varying the L/N ratio, we pass from a situation where wffs are underconstrained (and thus mostly satisfiable) to one where wffs are overconstrained (and thus mostly unsatisfiable). As a consequence, the plot of the satisfiability percentages draws a transition from 100% satisfiability to 100% unsatisfiability [MSL92]. Moreover, the mean and median CPU times plots reveal a easy-hard-easy pattern centered in the "100% satisfiable-100% unsatisfiable" transition zone. Increasing N the plots become sharper. Therefore, suitable choices for L and N allow us to generate very hard wffs with near 50% satisfiability probability.

In order to extend the fixed 3-clause-length test methodology to modal wffs, let us first consider the following definition of 3CNF modal wff.

- A 3CNF modal wff is a conjunction of 3CNF modal clauses.
- A 3CNF modal clause is a disjunction of three 3CNF modal literals, i.e., 3CNF modal variables or their negations.
- A 3CNF modal variable is either a propositional variable or a wff in the form $\Box C'$, where C' is a 3CNF modal clause.

This is a variant of the definition of modal CNF. Notice that conjunctions occur only at the top level. For instance, φ as in Example 1 is a 3CNF modal wff of depth 1. If the depth is 0, the definition collapses into propositional 3CNF. Random 3CNF modal wffs are generated by a procedure which generalizes that

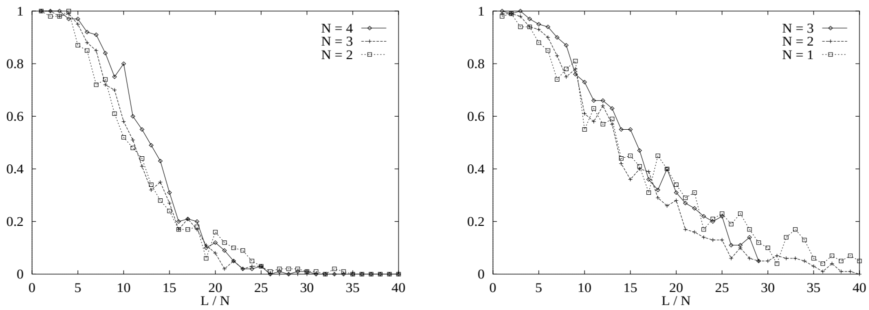

Fig. 3. Satisfiability percentages. 100 samples/point. Left: $p = 0.5$, $d = 1$, $N = 2, 3, 4$, $L/N = 1 \ldots 40$. Right: $p = 0.5$, $d = 2$, $N = 1, 2, 3$, $L/N = 1 \ldots 40$.

for 3SAT described above. This procedure takes in input the number of clauses L, the number of propositional variables N, the modal depth d and a probability value p. A 3CNF modal wff of depth d is produced by randomly generating L modal 3CNF clauses of depth d. A 3CNF modal clause of depth d is produced by randomly generating three 3CNF modal atoms of depth d, and negating each with probability 0.5. A 3CNF modal atom of depth d is produced in the following way. If $d = 0$, then one of the N propositional variables is returned randomly; otherwise either a random propositional variable (with probability p) or a box $\Box$ followed by a randomly generated 3CNF clause of depth $d - 1$ (with probability $1 - p$) is returned. (Intuitively, the parameter p establishes the mean ratio of the propositional atoms at every level of the wff tree.) The modal 3-clause-length test method is then defined in analogy with the propositional case. Given N, d and p, for increasing values of the clause number L, a certain number (100, 500, 1000...) of random modal 3CNF wffs are generated and given in input to the procedure under test. Satisfiability percentages and mean/median CPU times are plotted against the L/N ratio. Notice that, if we set $d = 0$, we have the standard 3SAT test method.

Similarly to the propositional case, the methodology proposed above presents three main features. First, it is easy to verify that there is a K-satisfiability-preserving way of converting modal wffs into modal 3CNF. (This can be done recursively on the depth of the wff, from the leaves to the root, each time applying to sub-wffs the propositional 3CNF conversion and the transformation $\Box \bigwedge_j \bigvee_{i=1}^{3} \varphi_{ij} \Longrightarrow \bigwedge_j \Box \bigvee_{i=1}^{3} \varphi_{ij}$.) Second, for any class of problems with depth d, there are only three parameters to handle, L, N and p. (This has been obtained by "lifting" all conjunctions to the top level of the wff.) Finally, the parameters L and N allow for a coarse "tuning" of both the satisfiability probability and the hardness of random 3CNF modal wffs. Figure 3 plots the satisfiability percentages against the L/N ratio obtained by an empirical test of 100 sample wffs/point, with $p = 0.5$, $d = 1, N = 2, 3, 4$ (left) and $d = 2, N = 1, 2, 3$ (right) respectively. Despite the noise due to the small number of samples, it can be noticed that, for any curve, by increasing L/N we pass monotonically from all random 3CNF wffs being satisfiable to a situation where they are all unsatisfiable. In Section 4 we show that the random wffs generated in this way are extremely hard.

4 Results

We have implemented all the four versions of KSAT for K(m) in Common Lisp in a very straightforward way, on top of a DPLL procedure for non-CNF wffs previously developed [AG93]. The function *assign* of Figures 1 and 2 performs a (linear time) lazy evaluation. The function *choose-literal(φ)* of Figures 1 and 2 performs the simple heuristic: "choose the variable with most occurrences inside φ". In the implementation of Version 4, *Likely-Unsatisfiable* is implemented trivially to always return "*True*", that is, the intermediate satisfiability check is always performed. If not otherwise specified, from now on we write "KSAT" to mean the version 4 of the algorithm described in Section 2.

In order to perform a comparative performance analysis of KSAT, we looked for available implemented procedures. There seem to be very few publicly available implemented decision procedures for K. The few ones we found are all tableau based, many of which only for S4 and S5. The tableau procedure we have used has been implemented in Common Lisp by Bernhard Nebel and lately improved by Enrico Franconi. It is a classical tableau procedure for K(m)/$\mathcal{ALC}$ [HM92, HNSS90], improved by a form of wff preprocessing. From now on we will refer to it simply as "Tableau" [4] [5].

Both Tableau and KSAT have been compiled on `AKCL 1.600` Common Lisp and executed under the `SunOS 4.1.3` operating system. In order to handle the huge amount of computation time required, we run our tests in parallel on 6 different SUN stations: (a) one `SPARC10 SUPERSPARC 32M`, (b) two twin `SPARC2 SUN4/75 32M`, (c) two twin `SPARC ELC SUN4/25 16M`, (d) one `SPARC SLC SUN4/20 16M`. For every problem group, Tableau and KSAT were always run on machines of the same configuration.

We compared Tableau and KSAT on six groups of test problems - labelled (i) to (vi) - generated randomly by the procedure described in Section 3, with $p = 0.5$, $d = 1$, $N = 2, 3, 4$ (groups (i) to (iii)) and $p = 0.5$, $d = 2$, $N = 1, 2, 3$ (groups (iv) to (vi)). Each group consists of a set of 4000 random wffs, organized into 40 subgroups of 100 sample wffs each, ranging from $L/N = 1$ to $L/N = 40$. This range was chosen empirically in a way to cover the transition between 100% satisfiability to 100% unsatisfiability described in Figure 3 of Section 3. All test wffs were previously ordered, along the lines described in Section 2.2. Figure 4 shows Tableau vs. KSAT mean CPU times (in seconds), plotted against the L/N ratio, for the six problem groups (i) to (vi) [6]. In both Figures 4 and 6, the labels (a)...(d) indicate the machine configuration on which each test was run. The necessity to perform the tableau tests in a reasonable time (e.g., the single

[4] KSAT, Tableau, the test code and all the results are available via anonymous FTP at `ftp.mrg.dist.unige.it` in `pub/mrg-systems/`.

[5] As the results presented in this section show, our system outperforms Tableau by orders of magnitude. The gap in performance is very big. At the moment it is not clear how much of the gap is due to the different technique, and how much is due to the (maybe different) quality of the two implementations.

[6] The highermost point of Tableau plot for problem group (iii) is computed only on 46 samples, as `AKCL` aborted for stack overflow at the 47th wff.

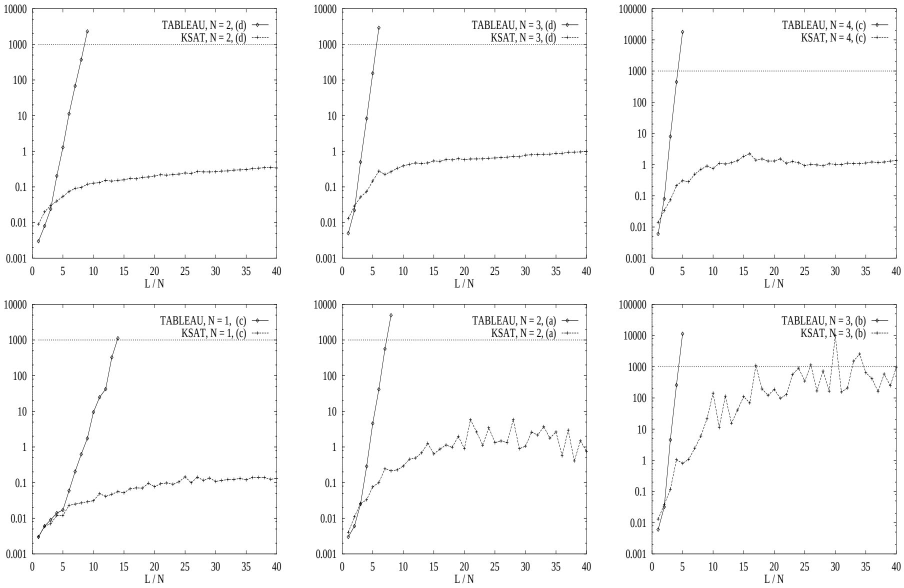

Fig. 4. Tableau vs. KSAT: mean CPU times (secs). 100 samples/point. First row, problem groups (i) to (iii): $p = 0.5$, $d = 1$, $N = 2, 3, 4$, $L/N = 1 \ldots 40$. Second row, problem groups (iv) to (vi): $p = 0.5$, $d = 2$, $N = 1, 2, 3$, $L/N = 1 \ldots 40$.

highermost point of the Tableau curve (v) required 137 hours CPU time on a SPARC10 machine!) imposed not to exceed the number of 100 samples/point, so that both mean CPU time and satisfiability ratio plots result rather noisy[7]. For the same reason, in all tests we had to stop the execution of Tableau whenever the mean CPU time on 100 samples exceeded a bound of 1000 seconds, which is indicated in Figures 4 and 6 by a horizontal line. Of course, whenever both procedures successfully ended computation, they always returned the same satisfiability values.

As it can be seen in Figure 4, KSAT drastically outperforms Tableau in all the problems considered (notice the logarithmic scale in the vertical axis). All Tableau mean CPU time plots present an exponential growth against the number of clauses, while KSAT mean CPU time plots grow much slowlier, seemingly polinomially (see below). The Tableau mean times curves reach the 1000 seconds time bound after very few steps, 10^3-10^4 times above the corresponding KSAT curves. The extrapolation of the Tableau curves suggests that the gap in performance may reach several orders of magnitude for problems near the right end side of the plots.

To provide a better understanding of the qualitative behaviour of KSAT, in Figure 5 (left) we have plotted both the mean and median CPU times resulting from re-running KSAT alone on problem group (iii), with 1000 sample wffs/point

[7] Similarly to what happens in SAT testing, the noise on KSAT mean CPU time plots is mainly due to single extremely-hard "outlier" problems. For instance, the peaks round 1000 seconds of KSAT plot (vi) are all due to single outliers, each requiring from 80,000 seconds to 900,000 seconds.

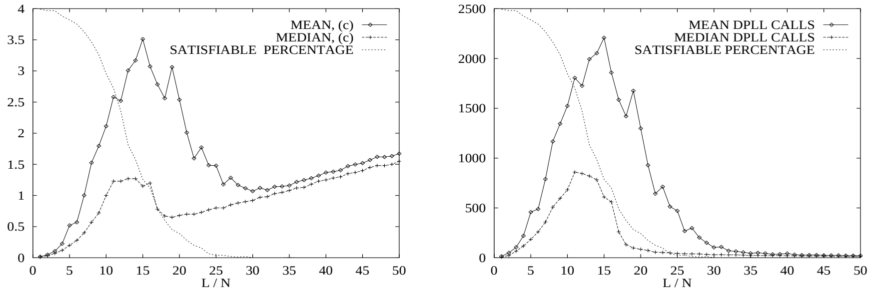

Fig. 5. KSAT: problem group (iii) with $L/N = 1 \ldots 50$, 1000 samples/point. Left: mean and median CPU times. Right: mean and median DPLL calls. Background: satisfiability percentages (100% satisfiable at end-scale).

and $L/N = 1 \ldots 50$. Notice that the vertical scale now grows linearly. The dotted line in the background represents the satisfiability percentage curve, the end of the scale beinging 100% satisfiability. (Notice the smoothness of the curve, with respect with the corresponding one in Figure 3). The plot reveals an easy-hard-easy pattern centered around the 50% crossover point, together with the linear component due to the linear-time-consuming function *assign*, which is invoked at any DPLL call. In Figure 5 (right) we have plotted the mean and median number of DPLL recursive calls (i.e., the size of the space searched), following what is standard practice in SAT testing (see, e.g., [MSL92]). It can be noticed that the linear component vanishes revealing only the easy-hard-easy pattern, analogously to what happens for SAT. The same qualitative behaviour is revealed by similar plots, not reported here, for problem groups (i), (ii) and (iv), while in problem groups (v) and (vi) the heavy noise hides any qualitative analysis.

In Figure 6 we compare the performances of Tableau, a modified version of Tableau (see below) and the different versions of KSAT - 2,3 and 4 - on problem group (v). (It is meaningless to consider Version 1 as the input wffs were ordered.) This allows us to give an overall and comparative evaluation of the various versions of the algorithm. As expected, already version 2 of KSAT outperforms Tableau. For instance, Tableau exceeds the time bound after 8 steps, about 10^2-10^3 times above the corresponding value of Version 2. Secondly, adding the factorization of $\bigwedge_i \alpha_i$ (Version 3) introduces a further improvement (about an order magnitude around the 13th step). Finally, the biggest improvement is obtained by adding partial assignment checking (Version 4). Again, the extrapolation of the incomplete curves suggests that the performance gaps might increase up to various orders of magnitude.

It may be argued that the bad performance of Tableau is due to Smullyan's propositional Tableau rules. [D'A92] proves that the $\vee$-rule may intrinsically cause inefficiency - as the two branches generated do not represent mutually inconsistent possible models - and proposes a modified set of rules which overcomes such a problem. [DM94] shows that the two main enhancements of Tableau rules used in automated deduction, *merging* and *lemma generation*, are made unnecessary by the modified rules. Most modal decision procedures implement standard analytic rules. Nevertheless, in order to provide a more general comparison, we

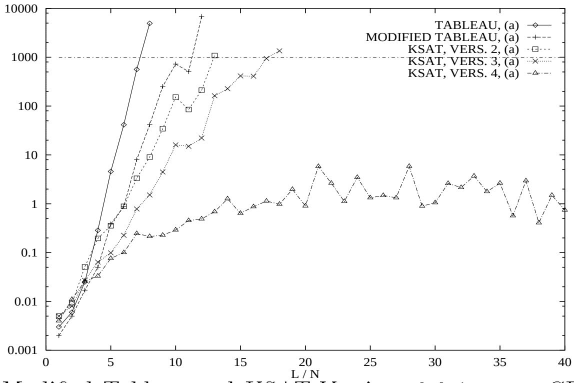

Fig. 6. Tableau, Modified Tableau and KSAT Versions 2,3,4: mean CPU times (secs). $d = 2$. $p = 0.5$. $N = 2$, $L/N = 1 \ldots 40$. 100 samples/point.

modified the code of Tableau according to the suggestions given in [D'A92] [8] and run the modified code on problem group (v). The resulting plot is reported in Figure 6 [9]. The experiment confirms D'Agostino's theoretical results, as the Modified Tableau outperforms the basic Tableau (e.g., a 10^2 factor at the 8th step). Despite this, the improvement introduced does not overcome the performance gap with KSAT, and the performance is still worse than that of Version 2 of KSAT.

5 Conclusions and future work

In this paper we have proposed and tested an algorithm for testing K-satisfiability which outperforms an existing tableau based decision procedure. The basic idea underlying this algorithm is that propositional modal decision procedures should be developed on top of propositional decision procedures.

This is clearly only a first step, and there are many directions for future research. The first, obvious one, is to improve on the heuristics. Some work in this direction has already been done and the preliminary results show that there is room for improvement. There are however other research issues which are more at the core of this research. An interesting issue is to what extent the kind of propositional reasoning used, and its efficiency, influence the efficiency of modal reasoning. Towards this direction we have already acquired and preliminarily tested what is considered one of the fastest implementation of DPLL, i.e., Max Böhm's system [BB92], and are in the process of acquiring implementations of OBDD, Dynamic Backtracking, and GSAT.

The other obvious direction of research is the extension to other modal logics. The current implementation works for K(m)/$\mathcal{ALC}$ and it has been adapted to work also for depth 1 S5, but no extensive testing has yet been done. We plan to consider other normal, non normal modal and terminological logics. It is our conjecture that this technique will be even more successful when applied to non normal modal logics.

---

[8] More specifically, we substituted the $\vee$-rule $\frac{\varphi \vee \psi}{\varphi \quad \psi}$ with the rules $\frac{\varphi \vee \psi}{\varphi \quad \psi, \neg\varphi}$, $\frac{\varphi \vee \psi}{\varphi, \neg\psi \quad \psi}$.

[9] The highermost point of Modified Tableau mean CPU time plot represents only 21 samples, as `AKCL` aborted for stack overflow at the 22nd sample wff.

References

[AG93] A. Armando and E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[BB92] M. Buro and H. Buning. Report on a SAT competition. Technical Report 110, University of Paderborn, Germany, November 1992.

[Bry92] R. E. Bryant. Symbolic Boolean manipolation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[D'A92] M. D'Agostino. Are Tableaux an Improvement on Truth-Tables? *Journal of Logic, Language and Information*, 1:235–252, 1992.

[DLL62] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[DM94] M. D'Agostino and M. Mondadori. The Taming of the Cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[Fit88] M. Fitting. First-Order Modal Tableaux. *Journal of Automated Reasoning*, 4:191–213, 1988.

[Gin93] M. L. Ginsberg. Dynamic Backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[GS94] F. Giunchiglia and L. Serafini. Multilanguage hierarchical logics (or: how we can do without modal logics). *Artificial Intelligence*, 65:29–70, 1994.

[GSGF93] F. Giunchiglia, L. Serafini, E. Giunchiglia, and M. Frixione. Non-Omniscient Belief as Context-Based Reasoning. In *Proc. IJCAI13*, pages 548–554, 1993.

[HM92] 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.

[HNSS90] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption Algorithms for Concept Description Languages. In *Proc. ECAI8*, pages 348–353, 1990.

[Mas94] F. Massacci. Strongly analytic tableaux for normal modal logics. In *Proc. CADE12*, 1994.

[MSL92] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proc. AAAI10*, pages 459–465, 1992.

[Sch91] K. D. Schild. A correspondence theory for terminological logics: preliminary report. In *Proc. IJCAI12*, pages 466–471, 1991.

[Seb94] R. Sebastiani. Applying GSAT to Non-Clausal Formulas. *Journal of Artificial Intelligence Research*, 1:309–314, 1994.

[SLM92] B. Selman, H. Levesque., and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proc. AAAI10*, pages 440–446, 1992.

[SS95] C. B. Suttner and G. Sutcliffe. The TPTP Problem Library. Technical Report TR 95/6, James Cook University, Australia, August 1995.

[SSS91] M. Schmidt-Schauß and G. Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48:1–26, 1991.

[US94] T. E. Uribe and M. E. Stickel. Ordered Binary Decision Diagrams and the Davis-Putnam Procedure. In *Proc. of the 1st International Conference on Constraints in Computational Logics*, 1994.