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.

[BFH+94] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

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

[GRS96] F. Giunchiglia, M. Roveri, and R. Sebastiani. A new method for testing decision procedures in modal and terminological logics. In *Proc. DL'96*, Cambridge, MA, USA, November 1996.

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

[GSGF93] F. Giunchiglia, L. Serafini, E. Giunchiglia, and M. Frixione. Non-Omniscient Belief as Context-Based Reasoning. In *Proc. IJCAI93*, pages 548–554, Chambery, France, 1993.

[HM92] J.Y. Halpern and Y. Moses. A guide to the completeness and complexity for modal logics of knowledge and belief. *Artif. Intel.*, 54(3):319–379, 1992.

[HNSS90] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption Algorithms for Concept Description Languages. In *Proc. ECAI90*, pages 348–353, 1990.

[Mas94] F. Massacci. Strongly analytic tableaux for normal modal logics. In *Proc. CADE94*, 1994.

[MSL92] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proc. AAAI92*, pages 459–465, 1992.

[Sch91] K. D. Schild. A correspondence theory for terminological logics: preliminary report. In *Proc. IJCAI91*, pages 466–471, Sydney, Australia, 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. AAAI92*, pages 440–446, 1992.

[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.

[WH94] C. P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Artif. Intel.*, 70:73–117, 1994.

This article was processed using the LaTeX macro package with LLNCS style

$L/N = 15 \sim 20$ in all the experiments. Moreover, in the first experiment a careful look reveals that the satisfiability transition becomes steeper when increasing N (e.g., compare the $N = 3$ and $N = 5$ plots). Finally, in all experiments, the curves representing the median number of DPLL calls (top row) locate the peaks inside the satisfiability transition, although they seem to anticipate a little the 50% crossover point.

From the above considerations we may conjecture (to be verified!) the existence for K(m) of a phase transition phenomenon, similar to that already known for SAT and other NP-hard problems (see, e.g, [MSL92, WH94]). As far as we know, this is the first time this phenomenon is revealed with modal formulas. This conjecture is also backed up by the analysis given in Section 4.3 (discussion on pruning unsatisfiable assignments). In fact, if we add a new clause/constraint to an unsatisfiable formula, this causes extra pruning in the search. Therefore, when we are in the 100% unsatisfiable zone, the search space size decreases with L/N. On the other hand, if we drop a clause/constraint from a satisfiable wff, this monotonically increases the number of satisfiable branches, and thus decreases the size of the search space visited to find a satisfiable branch. Therefore, when we are in the 100% satisfiable zone, the search space size increases with L/N. These two facts locate the peak inside the satisfiability transition zone. It is interesting to notice that, instead, the semantic branching has the effect of pushing down the performance curves for any value of L/N. In fact, no matter in which satisfiability zone we are, it always decreases the number of assignments considered.

6 Conclusions and future work

In this paper we have proposed and tested KSAT, an algorithm built on top of DPLL which tests K(m)-satisfiability. KSAT outperforms quantitatively and qualitatively the previous state-of-the-art tableau-based modal decision procedures, and has allowed us, among other things, to reveal what looks like a phase transition phenomenon for K(m).

This work is only an instance of the more general idea that propositional modal decision procedures should be developed on top of propositional decision procedures. There are clearly many directions for future research. An interesting issue is to what extent the kind of propositional reasoning used, and its efficiency, influence the efficiency of modal reasoning. For instance, we conjecture that all the modal procedures developed on top of propositional procedures which perform semantic branching and pruning will show a phase transition phenomenon. 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 an implementation of OBDDs.

The other obvious direction of research is the extension to other modal logics. We plan to consider other normal, non normal modal, temporal and dynamic logics. It is our conjecture that this technique will be even more successful when applied to non normal modal logics.

effectively searched by KSAT$_s$. [7]

The results in Figure 6 show how efficiency and satisfiability are affected by each single parameter. The results of the first experiment (left column) show that increasing N (and L accordingly) causes a relevant increase in complexity, up to one order of magnitude per variable in the "hard" zone. This should not be a surprise, as in K(m), adding few variables may cause an exponential increase of the search space. Each variable may in fact assume distinct truth values inside distinct states/possible worlds, that is, each variable must be considered with an "implicit multiplicity" equal to the number of states of a potential Kripke model. The results of the second experiment (center column) present two interesting aspects. First, the complexity of the search monotonically decreases with the increase of the number m of modalities (middle and bottom box). At a first sight it may sound like a surprise, but it should not be so. In fact, each truth assignment μ is partitioned into m independent sub-assignments μ_r's, each restricted to a single $\Box_r$ (see Equations (1) and (2)). This means "dividing and conquering" the search tree into m non-interfering search trees. Therefore, the bigger is m, the more partitioned is the search space, and the easier is the problem to solve. Second, a careful look reveals that the satisfiability percentage increases with m. Again, there is no mutual dependency between the satisfiability of the distinct μ_r's. Therefore the bigger is m, the less constrained is μ, and the more likely satisfiable is φ. The results of the third experiment (right column) provide evidence of the fact that the complexity increases with the modal depth d. This is rather intuitive: the higher is d, the deeper are the Kripke models to be searched, and the higher is the complexity of the search.

Giving a global look at the middle and bottom rows it can be noticed that these curves show the existence of an easy-hard-easy component plus a linear component. The former represents the number of recursive DPLL calls, i.e., the size of the tree effectively searched, while the latter is due to the preprocessing and to the linear-time function *assign*, which is invoked at every DPLL recursive call. In fact, if we increase N from 3 to 5 and L accordingly (left column), the size of the search space has a relevant increase. Therefore, while for $N = 3$ the linear component prevails, for $N = 5$ the easy-hard-easy component dominates. Moreover, when varying the number of modalities (center column), the wff sizes are kept the same for all curves. Therefore, when the effect of the easy-hard-easy component vanishes ($L/N > 20$), the curves collapse together, as the linear component does not depend on the number of modalities m. Notice that the locations of the easy-hard-easy zones do not seem to vary significantly, neither with the number of variables N (left column), nor with the number of modalities m (center column), nor with the depth d (right column). Let us now consider the satisfiability plots (top row). Despite the noise and the approximations due to timeouts, it is easy to notice that the 50% satisfiability point is centered around

[7] It can be noticed that the figures in the middle row report also the KRIS performance curves. These curves have been reported in order to back up the results and analysis in Section 4. As it can be noticed, KSAT$_s$ outperforms KRIS in all the testbeds, independently on the number of variables N, the number of modalities m or the depth d considered. Again, this is a quantitative and qualitative performance gap.

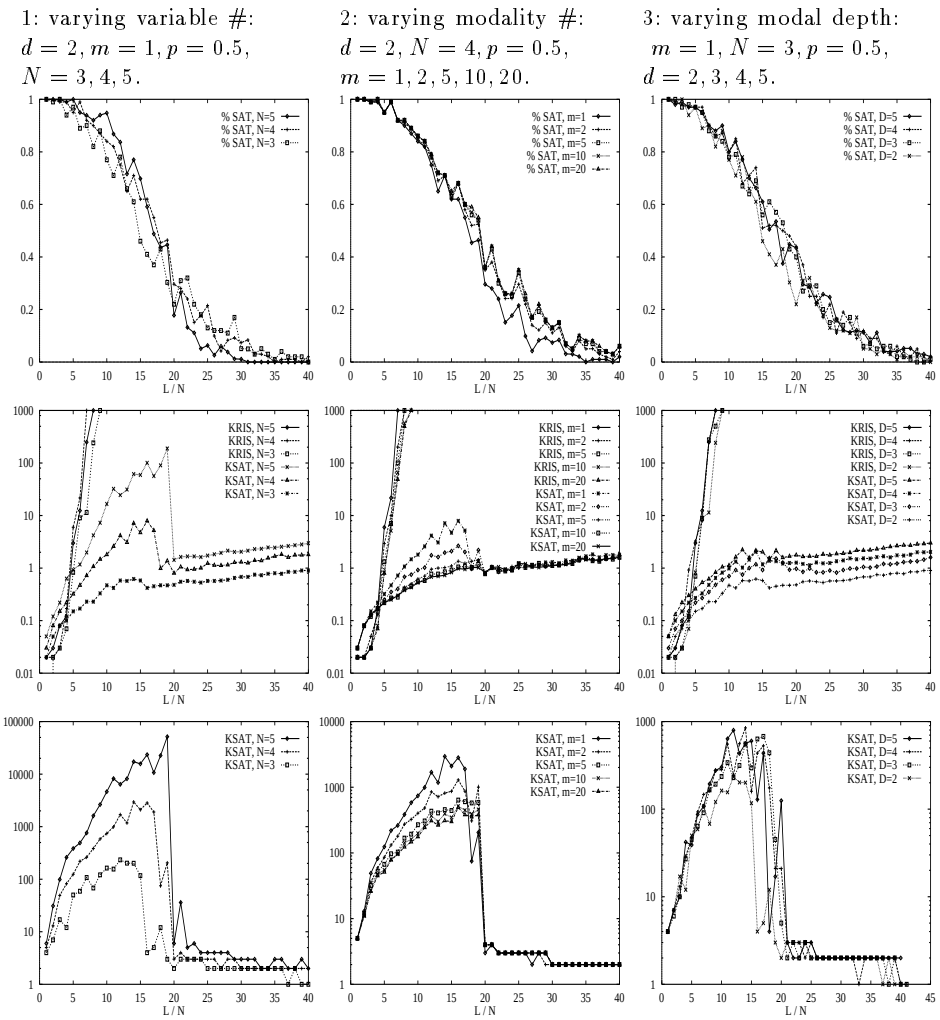

Fig. 6. The results of the three experiments.

1 (left column) we fix $d = 2$, $m = 1$, $p = 0.5$ and plot different curves for increasing numbers of the variables $N = 3, 4, 5$. In Experiment 2 (center column) we fix $d = 2$, $N = 4$, $p = 0.5$ and plot different curves for increasing numbers of distinct modalities $m = 1, 2, 5, 10, 20$. In Experiment 3 (right column) we fix $m = 1$, $N = 3$, $p = 0.5$ and plot different curves for increasing modal depths $d = 2, 3, 4, 5$. For each experiment, we present three sets of curves, each corresponding to a distinct row. In the first (top row) we plot the percentage of satisfiable wffs evaluated by KSAT$_s$. This gives a coarse indication of the average level of constraintness of the test wffs. In the second (middle row) we plot the median CPU time obtained by running both KSAT$_s$. This gives an overall picture of the KSAT$_s$qualitative behaviour. In the third (bottom row) we plot the KSAT$_s$ median number of recursive DPLL calls, that is, the size of the space

so forth. For instance, if $\alpha = \Box_1((\alpha' \vee \neg\beta') \wedge (\alpha' \vee \beta'))$ and $\beta = \Box_1(\alpha' \wedge \beta')$, then at the lower level we have a wff φ' of depth $d-1$ analogous to φ. This propagates exponentially the redundancy with the depth d. Finally, notice that if we considered the wff $\varphi^K = \bigwedge_{i=1}^{K}(\alpha_i \vee \neg\beta_i) \wedge (\alpha_i \vee \beta_i) \wedge (\neg\alpha_i \vee \neg\beta_i)$, the tableau would generate 2^K identical truth assignments $\mu^K = \bigwedge_i \alpha_i \wedge \neg\beta_i$, and things would get exponentially worse.

A DPLL-based procedure, instead, branches with $\alpha = True$ or $\alpha = False$. The first branch generates $\alpha \wedge \neg\beta$, while the second gives $\neg\alpha \wedge \neg\beta \wedge \beta$, which immediately closes. Therefore, only one instance of $\mu = \alpha \wedge \neg\beta$ is generated. The same applies recursively to μ^K. □

4.3.2 Detecting constraint violations

The explanation above leaves two questions unanswered. First, Figure 4 shows a performance gap between Version 2 and MODIFIED TABLEAU (more than a factor 30 for $L/N = 12$). This fact is still not explained, as MODIFIED TABLEAU generates mutually inconsistent subtrees (see Section 4.1). Second, Figure 5 (left) (but see also Figure 6 in Section 5) shows that, for L/N bigger than a certain value, the median time curves of KSAT decreases with the size of the formulas.

A propositional wff φ can be seen as a set of constraints for the truth assignments which possibly satisfy it (see, e.g., [WH94]). For instance, a clause $A_1 \vee A_2$ constrains every assignment not to set both A_1 and A_2 to F. Unlike tableau, DPLL prunes branches as soon as they violate some constraint of the input wff. The more constrained the input wff is, the more likely a truth assignment violates some constraint. (For instance, the bigger is L in a CNF wff, the more likely an assignment generates an empty clause.) Therefore, as φ becomes highly constrained (e.g., when L is big enough) the search tree is very heavily pruned. As a consequence, for L bigger than a certain value, the size of the search tree decreases with L.

Example 7. Consider the wff $\varphi = (\alpha \vee \varphi_1) \wedge (\beta \vee \varphi_2) \wedge \varphi_3 \wedge (\neg\alpha \vee \neg\beta)$, being α and β atoms, φ_1, φ_2 and φ_3 sub-wffs, such that $\alpha \wedge \beta \wedge \varphi_3$ is propositionally satisfiable. Again, assume the $\vee$-rule is applied in order, left to right. After two steps, the branch α, β is generated, which violates the constraint imposed by the last clause $(\neg\alpha \vee \neg\beta)$. A tableau-based procedure is not able to detect such a violation until it explicitly branches on that clause, that is, only after having generated the whole sub-tableau T' for $\alpha \wedge \beta \wedge \varphi_3$, which may be rather big. DPLL, instead, detects the violation and immediately prunes the branch. For instance, in KSAT this is done by the function *assign*. □

5 KSAT on K(m)

We have tested KSAT_s on a total of 48000 random $3\text{CNF}_{K(m)}$ wffs, organized in three experiments. As above, we have computed 100 samples/point, and chosen the range $\{1 \ldots 40\}$ for L/N to cover the "100% satisfiable – 100% unsatisfiable" transition. The results are reported in Figure 6. In each experiment we investigate the effects of varying one parameter while fixing the others. In Experiment

satisfiability of the assignments found. Both algorithms perform the latter step in the same way. The key difference is in the way KSAT and TABLEAU handle propositional inference. Tableau propositional decision procedures have, with respect to DPLL, two weaknesses which make them intrinsically less efficient and whose effects get (up to) exponentially amplified when using them in modal inference. Let us consider them in turn.

4.3.1 Syntactic vs. semantic branching

In a tableau propositional decision procedure truth assignments are (implicitly) generated as branches of an analytic propositional tableau. Analytic propositional tableaux perform what we call *syntactic branching*, that is, a branching on the syntactic structure of φ. In particular, as discussed in [DM94], an application of the $\vee$-rule (see Section 4.1) generates two subtrees which are *not mutually inconsistent*. The number of truth assignments generated grows exponentially with the number of disjunctions occurring positively in φ (in our tests, the number of clauses L). Therefore, the set of truth assignments enumerated by propositional tableau procedures is intrinsically redundant, and may contain (up to exponentially many) duplicated and/or subsumed assignments.

Things get much worse in the modal case. When testing K(m)-satisfiability, unlike the propositional case where tableaux look for *one* assignment satisfying the input formula, the propositional tableaux are used to enumerate all the truth assignments, which must be recursively checked for K(m)-consistency. Notice that the number of assignments can be huge: up to many thousands in our tests. This requires checking recursively (possibly many) sub-wffs of the form $\bigwedge_i \alpha_{ri} \wedge \beta_j$ of depth $d-1$, for which a propositional tableau will enumerate truth assignments, and so forth. Any redundant truth assignment enumerated at depth d introduces a redundant modal search tree of depth d. Even worse, this propositional redundancy propagates up to exponentially with the depth d, following the analysis of the sub-wffs of decreasing depth.

In DPLL-based procedures, instead, truth assignments are generated one-shot by DPLL. DPLL-based procedures perform a search based on what we call *semantic branching*, that is, a branching on the truth value of the proper sub-wffs of φ. Every branching step generates two *mutually inconsistent* subtrees. Because of this, DPLL always generates non-redundant sets of assignments. This avoids any search duplication and, in the case of modal search, any recursive exponential propagation of inefficiency.

Example 6. Consider the simple wff $\varphi = (\alpha \vee \neg\beta) \wedge (\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)$, where α and β are modal atoms, and let d be the depth of φ. The only possible assignment satisfying φ is $\mu = \alpha \wedge \neg\beta$. The $\vee$-rule is applied to the three clauses occurring in φ in the order they are listed, and two distinct but identical open branches are generated, both representing the assignment μ. Suppose now that μ is not K(m)-consistent. Then the tableau expands the two open branches in the same way, until it generates two identical (and possibly big) closed modal sub-tableaux T of depth d, each proving the K(m)-unsatisfiability of μ. This phenomenon may repeat itself at the lower level in each sub-tableaux T, and

implementation does not seem to affect the qualitative behaviour of the procedures. In fact, as far as we can see, both the TABLEAU and KRIS curves present an exponential growth with the number of clauses, while the KSAT and KSAT$_s$ curves flatten when the number of clauses exceeds a certain value. Third, independently from the quality of implementation, KSAT and KSAT$_s$ quantitatively outperform TABLEAU and KRIS. For instance, the performance gap between KSAT$_s$ and KRIS at the 10th step is about 4 orders of magnitude. Moreover, the extrapolation of the KRIS curve suggests that its value, and the performance gap with KSAT$_s$, would reach several orders of magnitude for problems at the right end side of the plots. To support this consideration, we have run KRIS on 100 samples of the same problem, for $L/N = 40$. No sample wff has been solved within the timeout. When releasing the timeout mechanism, KRIS has not been able to end successfully the computation of the first sample wff after a run of one month. Fourth, and most important, independently of the quality of implementation, KSAT and KSAT$_s$ qualitatively outperform TABLEAU and KRIS. In fact, while TABLEAU and KRIS present an exponential growth against the number of clauses, the KSAT and KSAT$_s$ curves present a polynomial growth (for more on this see Section 5).

To provide further evidence of the performance gap between DPLL-based procedures and tableau-based procedures, we have performed another, quite different, test, based on the class of wffs $\{\varphi_d^K\}_{d=1,2,\ldots}$ presented in [HM92]. This is a class of K(1)-satisfiable wffs, with depth d and $2d+1$ propositional variables. These wffs are paradigmatic for modal K, as every Kripke structure satisfying φ_d^K has at least $2^{d+1}-1$ distinct states, while $|\varphi_d^K|$ is $O(d^2)$. From the results in [HM92] we can reasonably assume a minimum exponential growth factor of 2^d for any ordinary algorithm based on Kripke semantics. We have run TABLEAU, KRIS, KSAT$_s$, KSAT Version 2, again compiled and run under `Allegro CL 4.2` on a `SUN SPARC10 32M` workstation, on these formulas, for increasing values of d. The results are plotted in Figure 5 (right). The TABLEAU, KRIS, KSAT$_s$ and KSAT Version 2 curves grow exponentially, approximatively as $(16.0)^d$, $(12.7)^d$, $(2.6)^d$ and $(2.4)^d$ respectively, exceeding 1000 seconds for $d = 6$, $d = 7$, $d = 11$ and $d = 12$ respectively. The slight difference between KSAT$_s$ and KSAT Version 2 is due to the overhead introduced by the $\bigwedge_i \alpha_{ri}$ factorization, which is useless with these formulas. It is worth observing that the result of tracing the global number of truth assignments μ, recursively found by both KSAT$_s$ and KSAT Version 2, has given exactly $2^{d+1}-1$ for every d, that is the minimum number of Kripke states. KSAT$_s$ and KSAT Version 2 have found no redundant truth assignments.

4.3 An explanation

The speed-up obtained by improving the quality of the implementation was to be expected. What is much more interesting is the quantitative and qualitative performance gap between the tableau-based procedures and the DPLL-based procedures. Let us concentrate on the basic algorithms and compare TABLEAU and KSAT. Both procedures work (i) by enumerating truth assignments which propositionally satisfy the input wff φ and (ii) by recursively checking the K(m)-

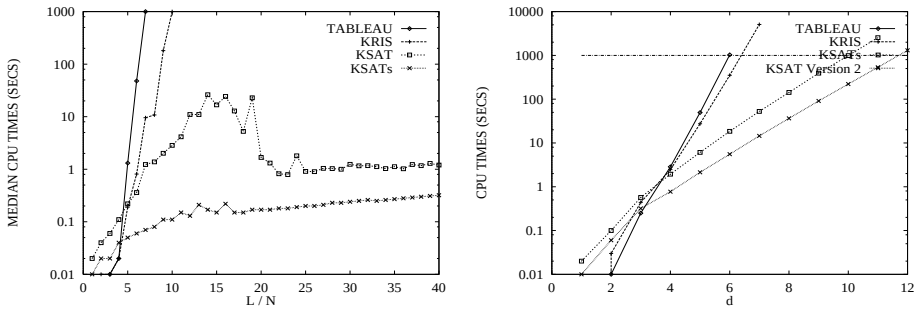

Fig. 5. Left: $d = 2$, $m = 1$, $N = 3$, $p = 0.5$, $L = N \ldots 40N$. TABLEAU, KRIS, KSAT and KSAT$_s$. Median CPU time, 100 samples/point. Right: TABLEAU, KRIS, KSAT$_s$ and KSAT Version 2 CPU times for φ_d^K formulas.

4.2 Comparing algorithms and systems

There are many tricks which can make an implementation more efficient. KSAT$_s$ has been obtained from KSAT by adding an initial phase of wff preprocessing and other relatively minor implementation variations (which can be understood simply by comparing the code of the two systems). [6] All the systems, that is TABLEAU, KRIS, KSAT and KSAT$_s$ have been compiled and run under `Allegro CL 4.2` on a `SUN SPARC10 32M` workstation. This has allowed us to use the builtin `Allegro` timeout mechanism.

We have compared TABLEAU, KRIS, KSAT and KSAT$_s$ on a testbed similar to group (vi) of Figure 3, that is, 4000 3CNF$_{K(m)}$ random formulas with $d = 2$, $m = 1$, $N = 3$, $p = 0.5$, $L/N \in \{1 \ldots 40\}$, with 100 samples/point. We have introduced some further improvements in the testing technique, again in order to minimize the testing time. First, we have introduced a timeout of 1000 seconds on each sample wff; any time the decision procedure under test has exceeded the timeout, the CPU time value has been conventionally set to 1000 seconds. Second, we have stopped running the test on a point whenever more than 50 samples have exceeded the timeout. Third, we have compared median values rather than mean values, as the former are much less sensitive to the noise introduced by outliers (see, e.g., [MSL92]). The results are presented in Figure 5 (left).

Four observations can be made, given below in increasing order of importance. First, improving the quality of the implementation, e.g., from TABLEAU to KRIS or from KSAT to KSAT$_s$, introduces good quantitative performance improvements. In fact, KRIS reaches the time bound at the 10th step, while TABLEAU reaches the time bound at the 7th step, about two orders of magnitude above the corresponding KRIS value. Similarly, KSAT has a maximum at the 14th step, more than 2 orders of magnitude above the corresponding KSAT$_s$ value. However, and this is the second observation, improving the quality of the

[6] The implementation of KSAT$_s$ is still naive in many respects, e.g., it is in Lisp and it does not use fancy optimized data structures. However the improvements allow us to get an idea of the effects of the quality of implementation on efficiency.

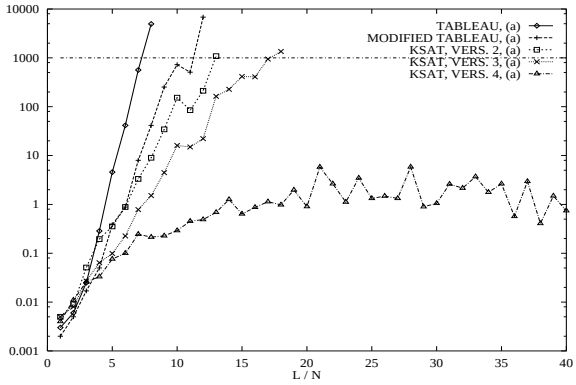

Fig. 4. TABLEAU, MODIFIED TABLEAU and KSAT Versions 2,3,4: mean CPU times (secs). $d = 2$. $p = 0.5$. $N = 2$, $L/N = 1 \ldots 40$. 100 samples/point.

every problem group, TABLEAU and KSAT have always been run under the same configuration. Second, we have not exceeded the number of 100 samples/point (giving a total of 4000 random wffs). Finally, in all the tests we have stopped the execution of TABLEAU whenever the mean CPU time on 100 samples has exceeded the bound of 1000 seconds. The results are reported in Figure 3. As it can be seen, KSAT outperforms TABLEAU in all the problems considered (notice the logarithmic scale in the vertical axis). All the TABLEAU mean CPU time plots present an exponential growth against the number of clauses, while the KSAT mean CPU time plots grow much slower. The TABLEAU curves reach the time bound of 1000 seconds after very few steps, 10^3-10^4 times above the corresponding KSAT curves. The extrapolation of the TABLEAU curves suggests that the gap may reach several orders of magnitude for problems near the right-end side of the plots.

In order to get an overall and comparative evaluation we have compared Versions 2,3 and 4 of KSAT and TABLEAU on problem group (v). (We could not consider Version 1 as the input wffs were sorted.) In this test we have also analyzed the behavior of MODIFIED TABLEAU, that is, TABLEAU modified to exploit *lemma generation* [DM94]. More specifically, MODIFIED TABLEAU is TABLEAU with the $\vee$-rule $\frac{\varphi \vee \psi}{\varphi \quad \psi}$ substituted with the rules $\frac{\varphi \vee \psi}{\varphi \quad \psi, \neg\varphi}$, $\frac{\varphi \vee \psi}{\varphi, \neg\psi \quad \psi}$. (The reasons for this choice will become clear in Section 4.3.) The results are reported in Figure 4. As it can be seen, already Version 2 outperforms both TABLEAU and MODIFIED TABLEAU. For instance, TABLEAU exceeds the time bound after 8 steps, about 10^2-10^3 times above the corresponding value of Version 2. MODIFIED TABLEAU outperforms TABLEAU (e.g., a 10^2 factor at the 8th step). Despite this, the improvement introduced does not overcome the performance gap with Version 2. Secondly, adding the factorization of $\bigwedge_i \alpha_i$ (Version 3) introduces a further improvement (about an order of magnitude around the 13th step). Finally, the biggest improvement is obtained by adding the incomplete assignment checking (Version 4). Again, the extrapolation of the curves suggests that the performance gaps might increase up to various orders of magnitude.

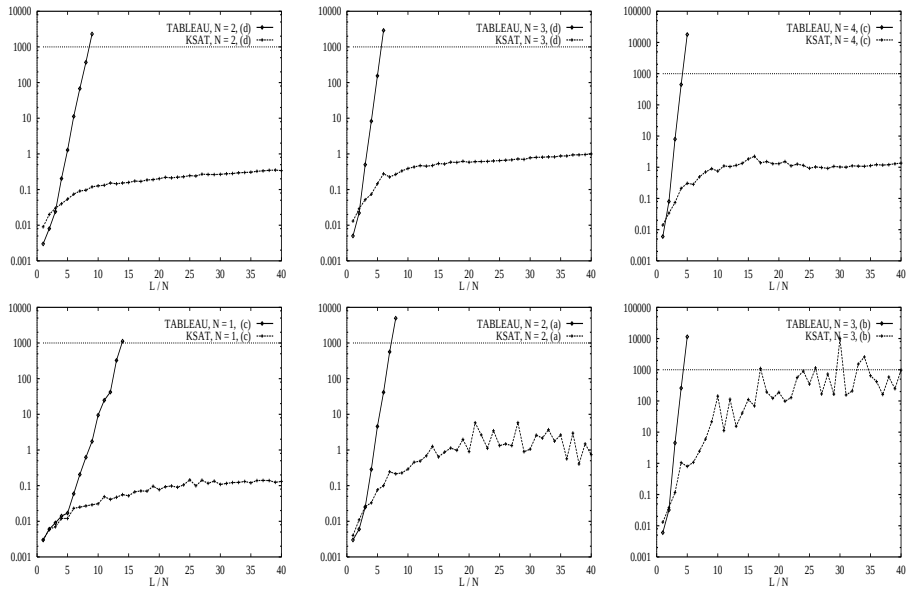

Fig. 3. TABLEAU vs. KSAT: mean CPU times (secs). 100 samples/point. First row, problem groups (i) to (iii): $p = 0.5$, $d = 1$, $N = 2, 3, 4$, $L/N = 1 \ldots 40$. Second row, problem groups (iv) to (vi): $p = 0.5$, $d = 2$, $N = 1, 2, 3$, $L/N = 1 \ldots 40$. The labels (a)...(d) indicate the machine configuration used for the test.

is randomly generated according to 5 parameters: (i) the modal depth d; (ii) the number of distinct boxes m; (iii) the number of top-level clauses L; (iv) the number of propositional variables N; (v) the probability p with which any random $3\text{CNF}_{K(m)}$ atom is propositional. For fixed N, d, m and p, for increasing values of L, many (100, 500, 1000...) random $3\text{CNF}_{K(m)}$ wffs are generated, internally sorted, and then given in input to the procedure under test. Satisfiability percentages, mean/median CPU times or mean/median search space sizes are plotted against the L/N ratio.

We have compared TABLEAU and KSAT on six groups of random $3\text{CNF}_{K(m)}$ wffs, labeled (i) to (vi), with $m = 1$, $p = 0.5$, $d = 1$, $N = 2, 3, 4$ (groups (i) to (iii)) and $m = 1$, $p = 0.5$, $d = 2$, $N = 1, 2, 3$ (groups (iv) to (vi)). Each group has been organized in 40 subgroups, each corresponding to an integer value of L/N ranging from 1 to 40. (As it will be clear in Section 5, this range has been chosen empirically to cover the transition between 100% satisfiability to 100% unsatisfiability). The necessity to perform the TABLEAU tests in a reasonable time (e.g., the single higher-most point of the TABLEAU curve (v) has required 137 hours of CPU time on a SPARC10 machine!) has imposed various constraints on the testing methodology. First, we have run our tests on 6 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`. However, for

disjunction of 3 $3\text{CNF}_{K(m)}$ literals, i.e., $3\text{CNF}_{K(m)}$ atoms or their negations; a $3\text{CNF}_{K(m)}$ atom is either a propositional atom or a wff of the form $\Box_r C$, where C is a $3\text{CNF}_{K(m)}$ clause.

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$_A$ calls per KSAT call could double, and the global number of KSAT$_A$ calls might increase of up to $2^{depth(\varphi)}$. To avoid this, it is worth introducing an heuristic function *Likely-Unsatisfiable*. The idea is that *Likely-Unsatisfiable* estimates the possibility of μ being K(m)-satisfiable 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 those we will see in next sections. Notice that, to make the intermediate consistency check worth doing, an average pruning of one single split per branch is sufficient.

4 DPLL-based vs. tableau-based decision procedures

We organize this section as follows. We start by comparing a direct implementation of the various versions of KSAT, as described in Section 4.1, with TABLEAU (unless explicitly stated the contrary, when we write KSAT, we mean KSAT Version 4). This allows us to compare the basic algorithms, without introducing "distortions" (i.e. speed-ups) due to smart implementation techniques. In Section 4.2 we compare TABLEAU, KRIS, KSAT and KSAT$_s$. This allows us to analyze the effects of implementation improvements on efficiency. We conclude by providing an explanation of the main results presented (Section 4.3). All the testing described in this section is restricted to the case of one modality. This allows us to consider the simplest situation without losing in generality; the phenomena highlighted in this section are in fact confirmed by the exhaustive testing presented in Section 5 (which studies, among other things, the effects of varying the number of modalities).

4.1 Comparing algorithms

We have implemented KSAT in Common Lisp on top of a DPLL procedure for non-CNF wffs previously developed [AG93]. With respect to the algorithm described in Section 3 we have made the following implementation choices: the function *assign* (see Figures 1 and 2) performs a (linear time) lazy evaluation; the function *choose-literal(φ)* (see 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 trivially implemented to always return "*True*". Both TABLEAU and KSAT have been compiled on `AKCL 1.600` and executed under `SunOS 4.1.3`.

To perform our tests, we have adopted the modal 3-clause-length test method, as described in [GRS96]. A particular kind of modal CNF wffs, called 3CNF$_{K(m)}$[5],

[5] A 3CNF$_{K(m)}$ wff is a conjunction of 3CNF$_{K(m)}$ clauses; a 3CNF$_{K(m)}$ clause is a

$$\bigwedge_i \alpha_{1i} = (\underline{\neg A_5} \vee A_4 \vee A_3) \ \wedge (\underline{\neg A_2} \vee \underline{\underline{A_1}} \vee A_4),$$
$$\neg\beta_{11} = A_3 \wedge A_1 \wedge \neg A_2,$$
$$\neg\beta_{12} = \neg A_4 \wedge A_2 \wedge \neg A_3.$$

Suppose *Incompatible-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 $\eta_1 \wedge \neg\beta_{11}$ is satisfiable but $\eta_1 \wedge \neg\beta_{12}$ is not. For the "base" step, *Incompatible-Subset* returns $B' = \{\beta_{12}\}$. Then *Incompatible-Subset* splits on the literal $\neg A_2$, finding $\eta_2 = \neg A_5 \wedge A_2 \wedge A_1$. As $\eta_2 \wedge \neg\beta_{12}$ is satisfiable, *Incompatible-Subset* returns the empty set. Therefore μ^1 is K(2)-satisfiable. □

3.4 Version 4: intermediate assignment checking

Despite the improvement brought by internally sorting modal atoms, the number of truth assignments found by KSAT$_W$ may still be too large. Version 4 starts from the empirical observation that most assignments found by KSAT$_W$ are "trivially" K(m)-unsatisfiable, that is, they will remain K(m)-unsatisfiable even after removing some of their conjuncts. If an incomplete [4] assignment μ' is K(m)-unsatisfiable, then all its extensions are K(m)-unsatisfiable. If the unsatisfiability of μ' is detected on time, then this prevents checking the K(m)-satisfiability of all the up to $2^{|TopAtoms(\varphi)|-|\mu'|}$ truth assignments which extend μ'.

This suggests the introduction of an intermediate K(m)-satisfiability test on incomplete 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$_W$ of Figure 1, just before the "split":

---

if *Likely-Unsatisfiable*(μ) /* intermediate assignment check */
  if not KSAT$_A$(μ)
    then return *False;*

Let us temporarily ignore the test performed by *Likely-Unsatisfiable*. KSAT$_A$ is invoked on the current incomplete assignment μ. If KSAT$_A$(μ) returns $False$, then all possible extensions of μ are unsatisfiable, and therefore KSAT$_W$ returns $False$.

Example 5. Consider the formula φ of Example 1. Suppose that, after three recursive calls, KSAT$_W$ builds the incomplete assignment:

$$\mu' = \Box_1(\neg A_1 \vee A_4 \vee A_3) \wedge \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge \neg\Box_1(A_4 \vee \neg A_2 \vee A_3)$$

(rows 6, 7 and 4 of φ). If it is invoked on μ', KSAT$_A$ will check the K(2)-satisfiability of the single formula

$$(\neg A_1 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \neg A_4 \wedge A_2 \wedge \neg A_3,$$

which is unsatisfiable. Therefore there will be no more need to select further literals, and KSAT$_W$ will backtrack. □

[4] By *incomplete* assignment μ for φ we mean that μ neither satisfies φ nor falsifies it.

```
function KSAT_RA(⋀_i □_r α_ri ∧ ⋀_j ¬□_r β_rj)
    if {there is no conjunct "¬□_r β_rj"}
        then return True;
    return Is-Empty(Incompatible-Subset(⋀_i α_ri, T, {β_r1, ..., β_rM_r}));

function Incompatible-Subset(α, η, B)
    if α = T                                                    /* base      */
        then return {β_rj ∈ B | KSAT(η ∧ ¬β_rj) = False};
    if α = F                                                    /* backtrack */
        then return B;
    if {a unit clause (l) is in α}                              /* unit      */
        then return Incompatible-Subset(assign(l, α), η ∧ l, B);
    l := choose-literal(α);                                     /* split     */
    B' := Incompatible-Subset(assign(l, α), η ∧ l, B) ;
    if Is-Empty(B')
        then return B';
        else return Incompatible-Subset(assign(¬l, α), η ∧ ¬l, B');
```

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

to KSAT_W, *Incompatible-Subset* tries to build truth assignments η's satisfying α. Whenever it finds one, all the wffs β_{rj}'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 $\eta \wedge \neg\beta_{rj}$, for all β_{rj}'s in B. The set of the wffs β_{rj}'s in B which are not compatible with η (i.e., $\text{KSAT}(\eta \wedge \neg\beta_{rj})$ returns $False$) is then returned.
- (backtrack) If $\alpha = F$, then η does not satisfy α. Therefore *Incompatible-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 *Incompatible-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 *Incompatible-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 more incompatible wffs are available, and thus B' is returned. Otherwise, *Incompatible-Subset* is invoked with *assign*$(\neg l, \alpha)$, $\eta \wedge \neg l$ and B'.

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

Example 4. Consider φ, μ^1, φ^{11} and φ^{12} as in Examples 1 and 2. We have

(in which case $\text{KSAT}_{RA}(\mu_r)$ returns *False*) or no more $\neg\Box_r\beta_{rj}$'s are available (in which case $\text{KSAT}_{RA}(\mu_r)$ returns *True*). Notice that KSAT_W, KSAT_A and KSAT_{RA} are a direct implementation of Theorems 1, 2 and 3, respectively. This guarantees their correctness and completeness.

3.2 Version 2: sorting modal atoms

One of the main causes of inefficiency of Version 1 is the number of truth assignments found by KSAT_W, which can be very large. This is a direct consequence of the large number of distinct modal atoms which can occur inside a K(m) 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) sorted, 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 3. Consider the modal atoms occurring in the wff φ in Example 1 (e.g., the atom $\Box_1(\neg A_3 \vee \neg A_1 \vee A_2)$ in the first row). For any atom in φ there may be up to $3! = 6$ equivalent permutations, which are all mapped into one atom (e.g., $\Box_1(\neg A_1 \vee A_2 \vee \neg A_3)$) if the modal atoms are sorted. □

3.3 Version 3: factorizing $\bigwedge_i \alpha_{ri}$

In Figure 1, KSAT_{RA} invokes repeatedly KSAT passing as arguments wffs of the form $\wedge_i\alpha_{ri} \wedge \neg\beta_{r1}$, ..., $\wedge_i\alpha_{ri} \wedge \neg\beta_{rM_r}$. All these wffs have the conjuncts $\wedge_i\alpha_{ri}$ in common. At the j-th call, KSAT searches for a K(m)-satisfiable truth assignment η_j satisfying $\wedge_i\alpha_{ri} \wedge \neg\beta_{rj}$. 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 $\wedge_i\alpha_{ri}$. The idea underlying Version 3 is to "factorize" the search of the truth assignments satisfying $\wedge_i\alpha_{ri}$. Given $\alpha = \wedge_i\alpha_{ri}$ and a non-empty set $B = \{\beta_{r1}, \ldots, \beta_{rM_r}\}$, 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_{rj}'s "compatible" with η_k (i.e., such that $\eta_k \wedge \neg\beta_{rj}$ is K(m)-satisfiable) are discharged from B. This is iterated till B is empty (μ^r is K(m)-satisfiable) or no more assignments η_k can be found (μ^r is not K(m)-satisfiable).

In Figure 2 we present a revised version KSAT_{RA}. As before, KSAT_{RA} takes in input a restricted truth assignment $\mu^r = \bigwedge_i \Box_r\alpha_{ri} \wedge \bigwedge_j \neg\Box_r\beta_{rj}$ and returns a truth value asserting whether μ^r is K(m)-satisfiable or not. If no conjunct of the form $\neg\Box_r\beta_{rj}$ occurs in μ^r, then μ^r is K(m)-satisfiable, and thus KSAT_{RA} returns *True*. Otherwise KSAT_{RA} invokes the function *Incompatible-Subset*, passing as arguments the wff $\alpha = \bigwedge_i \alpha_{ri}$, an empty assignment T, and the (always non-empty) wff set $B = \{\beta_{r1}, \ldots, \beta_{rM_r}\}$. *Incompatible-Subset*$(\alpha, T, B)$ returns the set of the wffs β_{rj}'s in B which are not compatible with any truth assignment which satisfies α. KSAT_{RA} returns *True* if and only if this set is empty. Similarly

function KSAT(φ)
  return KSAT$_W$(φ, T);

function KSAT$_W$(φ, μ)
  if $\varphi = T$ /* base */
    then return KSAT$_A$(μ);
  if $\varphi = F$ /* backtrack */
    then return *False*;
  if {a unit clause (l) occurs in φ} /* unit */
    then return KSAT$_W$($assign(l, \varphi), \mu \wedge l$);
  l := *choose-literal*(φ); /* split */
  return KSAT$_W$($assign(l, \varphi), \mu \wedge l$) **or**
    KSAT$_W$($assign(\neg l, \varphi), \mu \wedge \neg l$);

function KSAT$_A$($\bigwedge_i \Box_1 \alpha_{1i} \wedge \bigwedge_j \neg\Box_1 \beta_{1j} \wedge \ldots \wedge \bigwedge_i \Box_m \alpha_{mi} \wedge \bigwedge_j \neg\Box_m \beta_{mj} \wedge \gamma$)
  for each box index r **do**
    if not KSAT$_{RA}$($\bigwedge_i \Box_r \alpha_{ri} \wedge \bigwedge_j \neg\Box_r \beta_{rj}$)
      then return *False*;
  return *True*;

function KSAT$_{RA}$($\bigwedge_i \Box_r \alpha_{ri} \wedge \bigwedge_j \neg\Box_r \beta_{rj}$)
  for each conjunct "$\neg\Box_r \beta_{rj}$" **do**
    if not KSAT($\bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj}$)
      then return *False*;
  return *True*;

Fig. 1. The basic version of KSAT algorithm.

- (unit) If a literal l occurs in φ as a unit clause, then l must be assigned T. [3] To obtain this, KSAT$_W$ 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$_W$ is first invoked recursively with arguments *assign*(l, φ) and $\mu \wedge l$. If the result is negative, then KSAT$_W$ is invoked with arguments *assign*($\neg l, \varphi$) and $\mu \wedge \neg l$.

KSAT$_W$ is a variant of a non-CNF version of DPLL. Unlike DPLL (which returns *True*) whenever an assignment μ has been found, KSAT$_W$ invokes KSAT$_A$(μ). Essentially, DPLL is used to generate truth assignments, whose K(m)-satisfiability is recursively checked by KSAT$_A$. KSAT$_A$(μ) invokes KSAT$_{RA}$(μ_r) (where "$_{RA}$" stands for *R*estricted *A*ssignment) for any index r such that $\Box_r$ occurs in μ. This is repeated until either KSAT$_{RA}$ returns a negative value (in which case KSAT$_A$(μ) returns *False*) or no more $\Box_r$'s are available (in which case KSAT$_A$(μ) returns *True*). KSAT$_{RA}$(μ_r) invokes KSAT(φ^{rj}) for any conjunct $\neg\Box_r \beta_{rj}$ occurring in μ_r. Again, this is repeated until either KSAT returns a negative value

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

Theorem 1. A modal formula φ is K(m)-satisfiable if and only if there exists a K(m)-satisfiable truth assignment μ such that $\mu \models_p \varphi$.

Notice that this result is not committed to K(m), and it can be extended to any logic which gives a standard interpretation to the propositional connectives.

Theorem 2. The truth assignment μ of Equation (1) is K(m)-satisfiable if and only if the restricted truth assignment μ^r of Equation (2) is K(m)-satisfiable, for all $\Box_r$'s.

Theorem 3. The restricted assignment μ^r of Equation (2) is K(m)-satisfiable if and only if, for every $\neg\Box_r\beta_{rj}$ occurring in μ^r, it is K(m)-satisfiable the wff

$$\varphi^{rj} = \bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj}\,. \tag{3}$$

Theorem 1 reduces the K(m)-satisfiability of a formula φ to the K(m)-satisfiability of its truth assignments. Theorems 2 and 3 show how to reduce the latter to the K(m)-satisfiability of formulas of smaller depth. This process can be applied recursively, decreasing the depth of the formula considered at each iteration.

Example 2. In Example 1, consider the formula φ and the assignments μ, μ^1 and μ^2. μ propositionally satisfies φ, as it verifies one literal for every clause. Thus, for Theorem 1, μ is K(m)-satisfiable if and only if φ is K(m)-satisfiable. For Theorem 2 μ is K(m)-satisfiable if and only if both μ^1 and μ^2 are. For Theorem 3, μ^2 is trivially K(m)-satisfiable, as it contains no negated boxes, and μ^1 is K(m)-satisfiable if and only if each of the wffs

$\varphi^{11} = \bigwedge_i \alpha_{1i} \wedge \neg\beta_{11} = (\neg A_5 \vee A_4 \vee A_3) \ \wedge (\neg A_2 \vee A_1 \vee A_4) \ \wedge A_3 \wedge A_1 \wedge \neg A_2,$
$\varphi^{12} = \bigwedge_i \alpha_{1i} \wedge \neg\beta_{12} = (\neg A_5 \vee A_4 \vee A_3) \ \wedge (\neg A_2 \vee A_1 \vee A_4) \ \wedge \neg A_4 \wedge A_2 \wedge \neg A_3$

are K(m)-satisfiable. As they both are, then φ is K(m)-satisfiable. □

3 The Algorithm(s)

3.1 Version 1: the basic algorithm

The basic version of KSAT is reported in Figure 1. KSAT takes in input a modal propositional wff φ and returns a truth value asserting whether φ is K(m)-satisfiable or not. KSAT invokes KSAT_W (where "$_W$" stands for "Wff"), passing as arguments φ and the truth value T (i.e., by (1), the empty truth assignment). KSAT_W tries to build a K(m)-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(m)-satisfiable, then φ is K(m)-satisfiable. Therefore KSAT_W invokes $\text{KSAT}_A(\mu)$ (where "$_A$" stands for (truth) Assignment). KSAT_A returns a truth value asserting whether μ is K(m)-satisfiable or not.
- (backtrack) If $\varphi = F$, then μ can not be a truth assignment for φ. Therefore KSAT_W returns $False$.

is given by restricting μ to the set of atoms in the form $\Box_r\psi$, where $1 \leq r \leq m$.

Notationally, we use the Greek letters μ, η to represent truth assignments. Furthermore, from now on we often write a truth assignment as a formula:

$$\mu = \bigwedge_i \Box_1\alpha_{1i} \wedge \bigwedge_j \neg\Box_1\beta_{1j} \wedge \ldots \wedge \bigwedge_i \Box_m\alpha_{mi} \wedge \bigwedge_j \neg\Box_m\beta_{mj} \wedge \gamma, \qquad (1)$$

where the $\Box_r\alpha_{ri}$'s are the boxed atoms set to $True$, $\Box_r\beta_{ri}$'s are the boxed atoms set to $False$, and $\gamma = \bigwedge_{k=1}^{R} A'_k \wedge \bigwedge_{h=R+1}^{S} \neg A'_h$ is a conjunction of propositional literals. Notice that γ is always consistent. Similarly, we represent restricted assignments as:

$$\mu^r = \bigwedge_i \Box_r\alpha_{ri} \wedge \bigwedge_j \neg\Box_r\beta_{rj}\,. \qquad (2)$$

Furthermore, we say that an assignment (restricted assignment) is K(m)-satisfiable meaning that its corresponding formula (1) ((2)) is K(m)-satisfiable.

Example 1. Consider the following K(2) formula φ:

$$\begin{aligned}
\varphi = \{&\underline{\neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2)} \vee A_1 \vee A_5\} \wedge \\
&\{\underline{\neg A_2} \vee \neg A_5 \vee \Box_2(\neg A_2 \vee \neg A_4 \vee \neg A_3)\} \wedge \\
&\{\underline{A_1 \vee \Box_2(\neg A_4 \vee A_5 \vee A_2)} \vee A_2\} \wedge \\
&\{\neg\Box_2(A_4 \vee \neg A_3 \vee A_1) \vee \underline{\neg\Box_1(A_4 \vee \neg A_2 \vee A_3)} \vee \neg A_5\} \wedge \\
&\{\neg A_3 \vee A_1 \vee \underline{\Box_2(\neg A_4 \vee A_5 \vee A_2)}\} \wedge \\
&\{\underline{\Box_1(\neg A_5 \vee A_4 \vee A_3)} \vee \Box_1(\neg A_1 \vee A_4 \vee A_3) \vee \neg A_1\} \wedge \\
&\{\underline{A_1 \vee \Box_1(\neg A_2 \vee A_1 \vee A_4)} \vee A_2\}
\end{aligned}$$

Consider the following truth assignment μ, which sets to T the literals which are underlined:

$$\begin{aligned}
\mu = {} & \Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge \quad \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge && [\textstyle\bigwedge_i \Box_1\alpha_{1i}] \\
& \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \wedge \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) \wedge && [\textstyle\bigwedge_j \neg\Box_1\beta_{1j}] \\
& \Box_2(\neg A_4 \vee A_5 \vee A_2) \wedge && [\textstyle\bigwedge_i \Box_2\alpha_{2i}] \\
& \neg A_2. && [\gamma]
\end{aligned}$$

Notice that the two occurrences of $\Box_2(\neg A_4 \vee A_5 \vee A_2)$ in rows 3 and 5 of φ are both assigned $True$. μ gives rise to two restricted assignments μ^1 and μ^2:

$$\begin{aligned}
\mu^1 = {} & \Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge \quad \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge && [\textstyle\bigwedge_i \Box_1\alpha_{1i}] \\
& \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \wedge \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) && [\textstyle\bigwedge_j \neg\Box_1\beta_{1j}] \\
\mu^2 = {} & \Box_2(\neg A_4 \vee A_5 \vee A_2) && [\textstyle\bigwedge_i \Box_2\alpha_{2i}].
\end{aligned}$$

□

A truth assignment μ for φ *propositionally satisfies* φ, written $\mu \models_p \varphi$, if and only if it makes φ evaluate to $True$, that is, for all sub-formulas φ_1, φ_2 of φ:

$$\begin{aligned}
&\mu \models_p \varphi_1,\ \varphi_1 \in TopAtoms(\varphi) && \Longleftrightarrow \varphi_1 = True \in \mu; \\
&\mu \models_p \neg\varphi_1 && \Longleftrightarrow \mu \not\models_p \varphi_1; \\
&\mu \models_p \varphi_1 \wedge \varphi_2 && \Longleftrightarrow \mu \models_p \varphi_1 \textit{ and } \mu \models_p \varphi_2.
\end{aligned}$$

We say that a partial truth assignment μ *propositionally satisfies* φ if and only if all the assignments for φ which extend μ propositionally satisfy φ. For instance, if $\varphi = \Box_1\varphi_1 \vee \neg\Box_2\varphi_2$, then the partial assignment $\mu = \{\Box_1\varphi_1 = True\}$ is such that $\mu \models_p \varphi$. In fact, both $\{\Box_1\varphi_1 = True, \Box_2\varphi_2 = True\}$ and $\{\Box_1\varphi_1 = True, \Box_2\varphi_2 = False\}$ propositionally satisfy φ. From now on, if not otherwise specified, when dealing with propositional satisfiability we do not distinguish between assignments and partial assignments.

4 we test the versions of KSAT, as described in Section 3, a further improved version of KSAT, called KSAT$_s$ (where "s" stands for "smart") which implements some smart implementation tricks, and compare them with TABLEAU and KRIS. [2] This allows us to show and discuss the first three results hinted above. Finally, in Section 5, we analyze in detail the efficiency curves of KSAT. This analysis allows us to study the behavior of KSAT on K(m) and, among other things, to clearly identify the phase transition phenomenon (fourth result) mentioned above. Section 6 provides some conclusive remarks and describes the directions for future work.

2 The formal framework

We use here the standard modal logics syntax and Kripke semantics (see, e.g., [HM92]). In particular we use $\wedge, \vee, \neg, T, F, \supset, \equiv, \Box_1, \ldots, \Box_m$ as logical symbols, and A_1, A_2, ... as propositional variables. We use Greek letters $\alpha, \beta, \varphi, \ldots$ to denote K(m) formulas. We call *depth* of φ, written *depth(φ)*, the maximum number of nested modal operators in φ.

Let us call *atom* any formula that cannot be decomposed propositionally, that is, any formula whose main connective is not propositional. Examples of atoms are, A_1, $\Box_1(A_1 \vee \neg A_2)$ and $\Box_2(\Box_1 A_1 \vee \neg A_2)$. A *literal* is either an atom or its negation. Given a formula φ, an atom [literal] is a *top-level atom [literal]* for φ if and only if it occurs in φ and under the scope of no boxes. $TopAtoms(\varphi)$ is the set of the top level atoms of φ.

A *truth assignment* μ for a modal formula φ is a truth value assignment to all the top-level atoms of φ:

$$\begin{aligned}\mu = \{\ & \Box_1\alpha_{11} = True, \ldots, \Box_1\alpha_{1N_1} = True, \Box_1\beta_{11} = False, \ldots, \Box_1\beta_{1M_1} = False, \\ & \ldots \\ & \Box_m\alpha_{m1} = True, \ldots, \Box_m\alpha_{mN_m} = True, \Box_m\beta_{m1} = False, \ldots \\ & A'_1 = True, \ldots, A'_R = True, A'_{R+1} = False, \ldots, A'_S = False\},\end{aligned}$$

where, e.g., "$\Box_1\alpha_{11} = True$" should be read as "assign $True$ to $\Box_1\alpha_{11}$". $A'_i \in \{A_1, A_2 \ldots\}$, for every i. Identical atoms are always assigned identical truth values. A crucial property of truth assignments is that different atoms, e.g., $\Box_2(\varphi_1 \vee \varphi_2)$ and $\Box_2(\varphi_2 \vee \varphi_1)$ or, even, $\Box_2\varphi_1$ and $\Box_2(\varphi_1 \wedge \varphi_1)$ are treated differently and may thus be assigned different truth values. In this respect, notice that the $\Diamond_r$'s are not part of the language. This allows us to avoid assignments like, e.g., $\mu = \{\Box_r\alpha = True, \Diamond_r\neg\alpha = True, \ldots\}$ which are intrinsically inconsistent.

A *partial* truth assignment μ for φ is a truth value assignment to a proper subset of the top-level atoms of φ. If $\mu_2 \subseteq \mu_1$, then we say that μ_1 *extends* μ_2 and that μ_2 *subsumes* μ_1. A *restricted truth assignment*

$$\mu^r = \{\Box_r\alpha_{r1} = True, \ldots, \Box_r\alpha_{rN_r} = True, \Box_r\beta_{r1} = False, \ldots, \Box_r\beta_{rM_r} = False\}$$

[2] The various versions of KSAT, the test code and all the results presented in this paper are available via anonymous FTP at `ftp.mrg.dist.unige.it` in `pub/mrg-systems/ksat/`. TABLEAU is available at `ftp.mrg.dist.unige.it` in `pub/mrg-systems/tableau/`. KRIS is available at `ftp.dfki.uni-sb.de` in `/pub/tacos/KRIS/`.

[SLM92]. This allows us to exploit the huge amount of technology developed in this area, which is very advanced and well understood.

In this paper we concentrate on the satisfiability problem, not restricted to CNF, for modal K(m), that is K with m modalities; and use DPLL for testing propositional satisfiability (SAT). [1] The reasons for this choice are manyfold. First, K(m) is an interesting logic per se; for instance it is well known that K(m) is a notational variant of the terminological logic $\mathcal{ALC}$ [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]. Third, DPLL is the most widely known and studied propositional decision procedure, and also one of the most efficient [BB92] (but see also [US94]).

We have tested and compared various refinements of the algorithm we have developed, called KSAT, among themselves and also against the decision procedures and systems for modal logics we have been able to acquire. It turns out that all these implementations are tableau-based. In this paper we consider two of them. The first is a straightforward implementation of the algorithm described in [HNSS90], due to B. Nebel and E. Franconi. This procedure is called TABLEAU from now on. The second is the state-of-the-art system KRIS described in [HNSS90, BFH+94]. The testing confirms our original intuitions and also highlights some very interesting and unexpected phenomena. We can summarize our results as follows:

1. KSAT outperforms all the other implementations of orders of magnitude;
2. increasing the quality of the implementation produces an increase of performance. This increase is only quantitative and it does not change the shape of the performance curves;
3. tableau-based decision procedures are intrinsically less efficient than KSAT. This difference is quantitative but also, and more importantly, qualitative. The efficiency of tableau-based decision procedures keeps decreasing with the increase of the length of the input formulas (normalized to the number of propositional variables), while the efficiency of KSAT, after having decreased for a while, increases again;
4. KSAT produces what looks like a phase transition phenomenon [MSL92, WH94]. If the current (still partial) evidence is confirmed, this is the first time that this phenomenon, well known for SAT and other NP-hard problems, is found in modal logics.

This paper is structured as follows. In Section 2 we introduce formal framework, definitions and notation, and provide three simple but important results which motivate the algorithm, and directly imply its correctness and completeness. In Section 3 we describe the algorithm. The 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. In Section

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

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

Fausto Giunchiglia[1,2] Roberto Sebastiani[1]

[1] IRST, 38050 Povo (TN), Italy. ph: ++39.461.314517
[2] DISA, via Inama 5, 38100 Trento, Italy.
{fausto,rseba}@irst.itc.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 consider satisfiability in modal K(m), that is modal K with m modalities, and develop an algorithm, called KSAT, on top of an implementation of the Davis-Putnam-Longemann-Loveland procedure. KSAT is thoroughly tested and compared with various procedures and in particular with the state-of-the-art tableau-based system KRIS. The experimental results show that KSAT outperforms KRIS and the other systems of orders of magnitude, highlight an intrinsic weakness of tableau-based decision procedures, and provide partial evidence of a phase transition phenomenon for K(m).

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 (see, e.g., [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], KE [DM94], or even partial decision procedures like GSAT

** A much longer version of this paper, containing the proofs of the theorems and more detailed examples and explanations, is the IRST technical report 9611-06, which is available via anonymous FTP at ftp.mrg.dist.unige.it in pub/mrg-ftp/.

* This work has benefited from many long discussions with Enrico Giunchiglia. Enrico played a crucial role in the development of Version 1 of KSAT. Marco Roveri has given technical assistance in the testing phase.