InKreSAT: Modal Reasoning via Incremental Reduction to SAT

Mark Kaminski[1] and Tobias Tebbi[2]

[1] Department of Computer Science, University of Oxford, UK
[2] Saarland University, Saarbrücken, Germany

Abstract. InKreSAT is a prover for the modal logics K, T, K4, and S4. InKreSAT reduces a given modal satisfiability problem to a Boolean satisfiability problem, which is then solved using a SAT solver. InKreSAT improves on previous work by proceeding incrementally. It interleaves translation steps with calls to the SAT solver and uses the feedback provided by the SAT solver to guide the translation. This results in better performance and allows to integrate blocking mechanisms known from modal tableau provers. Blocking, in turn, further improves performance and makes the approach applicable to the logics K4 and S4.

1 Introduction

InKreSAT is a prover for the modal logics K, T, K4, and S4 [3] that works by encoding modal formulas into SAT. The idea of a modal prover based on SAT encoding has previously been explored by Sebastiani and Vescovi [15]. While building on the same basic idea, InKreSAT extends the approach in [15] in several ways. Rather than encoding the entire modal formula in one go and then running a SAT solver on the resulting set of clauses, InKreSAT interleaves encoding phases with calls to an incremental SAT solver. If the SAT solver returns unsatisfiable, the initial modal problem is unsatisfiable, so no further encoding needs to be done. Otherwise, the SAT solver returns a propositional model of a partial encoding of the modal formula, which is used by InKreSAT to guide further encoding steps. While InKreSAT is the first system that decides modal satisfiability by incremental encoding into SAT, similar ideas have been explored for semi-decision procedures for first-order [7] and higher-order logic [4].

To deal with transitivity as it occurs in K4 and S4, and to further improve the overall performance of InKreSAT, we extend our basic approach by blocking (see, e.g., [11]).

We evaluate InKreSAT, confirming the effectiveness of our incremental approach compared to Sebastiani and Vescovi's one-phase approach. InKreSAT also proves competitive with state-of-the-art modal tableau provers.

InKreSAT is implemented in OCaml and employs the SAT solver MiniSat [5] (v2.2.0). The source code of InKreSAT and the benchmark problems used in the evaluation are available from www.ps.uni-saarland.de/~kaminski/inkresat.

2 Reduction to SAT

We now present the SAT encoding that underlies InKreSAT. We restrict ourselves to the case of multimodal K. An alternative, more detailed presentation of (a variant of) the encoding can be found in [15].

We distinguish between *propositional variables* (denoted p, q) and *relational variables* (denoted r). From these variables, the *formulas* of K can be obtained by the following grammar: $\varphi, \psi \ ::= \ p \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle r\rangle\varphi \mid [r]\varphi$.
We call formulas of the form $\langle r\rangle\varphi$ *diamonds* and formulas of the form $[r]\varphi$ *boxes*.

We assume a countably infinite set of *prefixes* (denoted σ, τ) and a strict total order $\prec$ on prefixes. We call pairs $\sigma : \varphi$ *prefixed formulas*. We assume an injective function that maps every prefixed diamond $\sigma : \langle r\rangle\varphi$ to a prefix $\tau_{\sigma:\langle r\rangle\varphi}$ such that $\sigma \prec \tau_{\sigma:\langle r\rangle\varphi}$. The SAT encoding underlying InKreSAT is based on the following tableau calculus for K (working on formulas in negation normal form).

$$(\neg)\ \frac{\sigma : \varphi,\ \sigma : {\sim}\varphi}{\otimes} \qquad (\wedge_i)\ \frac{\sigma : \varphi_1 \wedge \varphi_2}{\sigma : \varphi_i}\ i \in \{1, 2\} \qquad (\vee)\ \frac{\sigma : \varphi_1 \vee \varphi_2}{\sigma : \varphi_1 \mid \sigma : \varphi_2}$$

$$(\Diamond)\ \frac{\sigma : \langle r\rangle\varphi}{\tau_{\sigma:\langle r\rangle\varphi} : \varphi} \qquad (\Box)\ \frac{\sigma : [r]\varphi,\ \sigma : \langle r\rangle\psi}{\tau_{\sigma:\langle r\rangle\psi} : \varphi}$$

In the formulation of $(\neg)$, we write ${\sim}\varphi$ for the negation normal form of $\neg\varphi$, while the symbol $\otimes$ stands for the empty conclusion that closes a branch.

It can be shown that the tableau calculus is sound and complete with respect to the relational semantics of K (in fact, the calculus yields a decision procedure for K). In other words, a formula φ of K is satisfiable if and only if there is a maximal tableau rooted at $\sigma : \varphi$ (for an arbitrary prefix σ) that has an open branch (we use the terms "tableau" and "(tableau) branch" as in [6]).

Literals (denoted l) are possibly negated propositional variables. We define $\overline{\neg p} := p$ and $\overline{p} := \neg p$. We assume an injective function that maps every prefixed formula $\sigma : \varphi$ to a literal $l_{\sigma:\varphi}$ such that $l_{\sigma:\varphi} = \bar{l}_{\sigma:\sim\varphi}$. Note that all of the above rules have the form $\frac{\sigma:\varphi_1,\ldots,\sigma:\varphi_m}{\tau:\psi_1\mid\cdots\mid\tau:\psi_n}$ where $m \in \{1, 2\}$ and $n \in \{0, 1, 2\}$. Thus, we can use the following mapping to assign a clause to every instance of a rule.

$$\frac{\sigma : \varphi_1, \ldots, \sigma : \varphi_m}{\tau : \psi_1 \mid \cdots \mid \tau : \psi_n} \quad \rightsquigarrow \quad \bar{l}_{\sigma:\varphi_1} \vee \cdots \vee \bar{l}_{\sigma:\varphi_m} \vee l_{\tau:\psi_1} \vee \cdots \vee l_{\tau:\psi_n}$$

The mapping can be lifted to tableaux as demonstrated by the following example:

Tableau		Rule		Clause
$\sigma : (p \vee q) \wedge \neg p$			$\rightsquigarrow$	$l_{\sigma:(p\vee q)\wedge\neg p}$
$\sigma : p \vee q$		$(\wedge_1)$	$\rightsquigarrow$	$\bar{l}_{\sigma:(p\vee q)\wedge\neg p} \vee l_{\sigma:p\vee q}$
$\sigma : \neg p$		$(\wedge_2)$	$\rightsquigarrow$	$\bar{l}_{\sigma:(p\vee q)\wedge\neg p} \vee l_{\sigma:\neg p}$
$\sigma : p$	$\sigma : q$	$(\vee)$	$\rightsquigarrow$	$\bar{l}_{\sigma:p\vee q} \vee l_{\sigma:p} \vee l_{\sigma:q}$
$\otimes$		$(\neg)$	$\rightsquigarrow$	$\bar{l}_{\sigma:p} \vee \bar{l}_{\sigma:\neg p}$ (redundant)

Note that the prefixed formula $\sigma : (p \vee q) \wedge \neg p$ at the root of the tableau is mapped to a unit clause since it is considered an assumption rather than a consequence of a tableau rule application. The last clause, which corresponds

Input: a formula φ_0
Variables: Agenda := $\{\sigma_0 : \varphi_0\}$ (for some arbitrary but fixed prefix σ_0)
while Agenda $\neq \emptyset$ do:
1. for all $\sigma : \varphi \in$ Agenda do:
   if φ is a diamond then add $C_\sigma\varphi \cup \{ B_\sigma\psi\varphi \mid \sigma : \psi$ processed, ψ box $\}$ to SAT solver
   else if φ is a box then add $\{ B_\sigma\varphi\psi \mid \sigma : \psi$ processed, ψ diamond $\}$ to SAT solver
   else if φ is a conjunction then add $C^1_\sigma\varphi$ and $C^2_\sigma\varphi$ to SAT solver
   else add $C_\sigma\varphi$ to SAT solver
2. run SAT solver
3. if SAT solver returns unsat then return unsat
   else Agenda := $\{ \sigma : \varphi \mid \sigma : \varphi$ pending, $l_{\sigma:\varphi}$ true in model returned by SAT solver $\}$

return sat

Fig. 1. InKreSAT: basic algorithm

to the application of $(\neg)$ to $\sigma : p$ and $\sigma : \neg p$, is redundant since our mapping of prefixed formulas to literals already ensures that $l_{\sigma:p}$ and $l_{\sigma:\neg p}$ are contradictory.

Thus, every tableau can be mapped to a set of Boolean clauses. It can be shown that the set is satisfiable if and only if the tableau has an open branch (a variant of the claim is shown in [15]). The encoding can be extended to T, K4, and S4 by suitably extending the underlying tableau calculus (see [6]).

3 Basic Algorithm

The basic algorithm underlying InKreSAT interacts with an incremental SAT solver by adding new clauses to the solver and periodically running the solver on the clauses added so far. If the solver returns satisfiable, it also returns a satisfying model that is used in selecting new clauses to be added.

The *premise of a clause* C, where C corresponds to an instance of a tableau rule $\frac{\sigma:\varphi_1,\ldots,\sigma:\varphi_m}{\tau:\psi_1|\cdots|\tau:\psi_n}$, consists of the literals $l_{\sigma:\varphi_1}, \ldots, l_{\sigma:\varphi_m}$. We call a prefixed formula $\sigma : \varphi$ *processed* if $l_{\sigma:\varphi}$ occurs in the premise of a clause added to the SAT solver. Otherwise, we call $\sigma : \varphi$ *pending*, but only if (1) $l_{\sigma:\varphi}$ occurs in a clause added to the SAT solver and (2) φ is not of the form p or $\neg p$. We exclude formulas $\sigma : p$ and $\sigma : \neg p$ because they require no further processing: since our mapping of prefixed formulas to literals takes care of trivial inconsistencies, we never generate clauses for the rule $(\neg)$.

Let φ be a disjunction or a diamond. We write $C_\sigma\varphi$ for the clause corresponding to the instance of $(\vee)$ or $(\Diamond)$, resp., that has $\sigma : \varphi$ as its unique premise (e.g., $C_\sigma(\langle r\rangle p) = \bar{l}_{\sigma:\langle r\rangle p} \vee l_{\tau_{\sigma:\langle r\rangle p}:p}$). If φ is a conjunction, we write $C^i_\sigma\varphi$ $(i \in \{1, 2\})$ for the clause corresponding to the instance of $(\wedge_i)$ that has $\sigma : \varphi$ as its premise. Finally, we write $B_\sigma\varphi\psi$, where φ is a box and ψ a diamond, for the clause corresponding to the instance of $(\Box)$ that has $\sigma : \varphi$ and $\sigma : \psi$ as its premises.

The basic algorithm (restricted to K) is shown in Fig. 1. It maintains an agenda consisting of pending prefixed formulas that are true in the model returned by a preceding invocation of the SAT solver (initially, the agenda contains the input formula). Every formula $\sigma : \varphi$ on the agenda is processed by adding clauses with $l_{\sigma:\varphi}$ in their premise (after which $\sigma : \varphi$ becomes processed).

4 Blocking

Blocking is a technique commonly used with tableau calculi to achieve termination in the presence of transitive relations or background theories [11]. Even when it is not required for termination, blocking can improve the performance of tableau-based decision procedures [10]. Blocking typically restricts the applicability of the rule ($\Diamond$). An application of ($\Diamond$) to a prefixed diamond on a branch is blocked if one can determine that the successor prefix that would be introduced by the application is subsumed by some prefix that is already on the branch.

Because of the close correspondence between the translational method underlying InKreSAT and modal tableau calculi, blocking is necessary to make our translation terminating in the presence of transitive relations.

Extending the one-phase approach in [15] with blocking is problematic since the approach has no explicit representation of tableau branches. Known blocking techniques are all designed to work on a single tableau branch at a time. Blocking across branches typically destroys the correctness of a tableau system.

In our case, however, the propositional model used to guide the translation in step 3 of the main loop in Fig. 1 yields a suitable approximation of a tableau branch—the formulas whose corresponding literals are true in the model. We can show that blocking restricted to these formulas preserves the correctness of our procedure. To explore the impact of blocking, we extend the basic algorithm by a variant of anywhere blocking [1] (with ideas from pattern-based blocking [12]).

Unlike with tableau provers [10], blocking in InKreSAT can cause considerable overhead. After every run of the SAT solver, the data structures needed for blocking may have to be recomputed from scratch because models returned by two successive runs of the solver may differ in an unpredictable way. To avoid the recomputation, we must be able to guarantee that the model returned by the solver is an extension of the previously computed model. This leads us to a final refinement of our procedure, called *model extension* (MX). We make the SAT solver always first search for extensions of the existing model by adding all literals true in the model to the input of the solver (as unit clauses). If the solver finds an extension of the model, we proceed without recomputing the data structures for blocking. Otherwise, we run the solver once again, now without the additional clauses, and recompute the data structures from scratch. The goal of MX is to reduce the overhead of blocking, thus increasing its effectiveness. On the other hand, MX can cause more calls to the solver, which may decrease performance.

5 Evaluation and Conclusions

We evaluate the effects of incremental translation to SAT and blocking by running InKreSAT in four different modes: a “one phase” mode, where, like in [15], the encoding is generated in one go, a “no blocking” mode, where clause generation is performed incrementally, but blocking is switched off, a “no MX” mode, where blocking is enabled, but MX is disabled, and the default mode, where both blocking and MX are enabled. Besides, we include the results from four other

Subclass	InKreSAT (default)	InKreSAT (no MX)	InKreSAT (no blocking)	InKreSAT (one phase)	Spartacus	FaCT++	K2SAT	*SAT
branch_n	12	12	13	4	9	10	**15**	12
branch_p	**18**	15	14	4	10	9	16	**18**
d4_n	**21**	**21**	7	6	**21**	**21**	6	**21**
d4_p	**21**	**21**	13	8	**21**	**21**	9	**21**
dum_n	**21**	**21**	**21**	17	**21**	**21**	19	**21**
dum_p	**21**	**21**	**21**	16	**21**	**21**	18	**21**
lin_n	**21**	**21**	**21**	**21**	**21**	**21**	**21**	13
path_n	14	**21**	6	6	**21**	**21**	13	**21**
path_p	12	**21**	8	7	**21**	**21**	14	**21**
ph_n	**21**	**21**	**21**	**21**	**21**	12	**21**	11
ph_p	**9**	**9**	**9**	**9**	8	7	**9**	8
t4p_n	**21**	**21**	7	4	**21**	**21**	4	**21**
t4p_p	**21**	**21**	13	8	**21**	**21**	8	**21**

Subclass	InKreSAT (default)	InKreSAT (no MX)	Spartacus	FaCT++
branch_n	**11**	**11**	9	6
md_n	8	9	**21**	10
md_p	3	4	**9**	4
ipc_n	9	11	**21**	10
ipc_p	8	11	**21**	9
path_n	4	9	16	**21**
path_p	5	10	17	**21**
ph_n	**13**	11	10	8
ph_p	**9**	**9**	5	6
s5_n	14	**20**	16	19

Table 1. Results on the LWB benchmarks for K (left) and S4 (right)

provers. (1) K2SAT, Sebastiani and Vescovi's [15] implementation of their one-phase translational approach. We used K2SAT in conjunction with MiniSat 2.0, which is directly integrated into the system (the integrated solution outperformed a setup using MiniSat 2.2.0, which is used by InKreSAT). We used the options `-j -u -v -w` recommended by the authors. (2) *SAT [16] (v1.3), a reasoner for the description logic $\mathcal{ALC}$. *SAT also integrates SAT technology, but does so in a way that is different from our approach. It uses a SAT solver only for propositional reasoning, while modal reasoning is handled by a conventional tableau calculus. (3) FaCT++ [17] (v1.6.1), an established reasoner for the web ontology language OWL 2 DL. (4) Spartacus [10] (v1.1.3), an efficient prover for the hybrid logic $\mathcal{H}(E, @)$. K2SAT and *SAT are included because they implement related approaches while FaCT++ and Spartacus are supposed to indicate the state of the art in automated reasoning for modal logic. Except for K2SAT, all provers are compiled and run with the default settings (unlike in [8]).

We perform the tests on a Pentium 4 2.8 GHz, 1 GB RAM, with a 60s time limit per formula (the same setup as in [10]). **Table 1:** The K and S4 problem sets from the Logic Work Bench (LWB) benchmarks [2]. LWB is widely used for measuring the performance of modal reasoners (e.g., in [8,10,15]). LWB is the only suite available to us that includes S4 problems. For each subclass that was not solved in its hardest instance (21) by every system, Table 1 displays the hardest instance that could be solved (the best results set in bold). The evaluation on the S4 problems is limited to systems and configurations of InKreSAT that can cope with transitivity. **Fig. 2**, upper half**:** Randomly generated 3CNF$_K$ [8] formulas of modal depth 2, 4, and 6 (45 problems each, 135 in total; see [10] for details). The selection allows us to see how performance depends on modal depth. We plot the number of instances that could be solved against time. The plot on the left-hand side compares the four different modes of InKreSAT, while on the right we compare InKreSAT to the other provers. **Fig. 2**, lower half**:** A subset of the

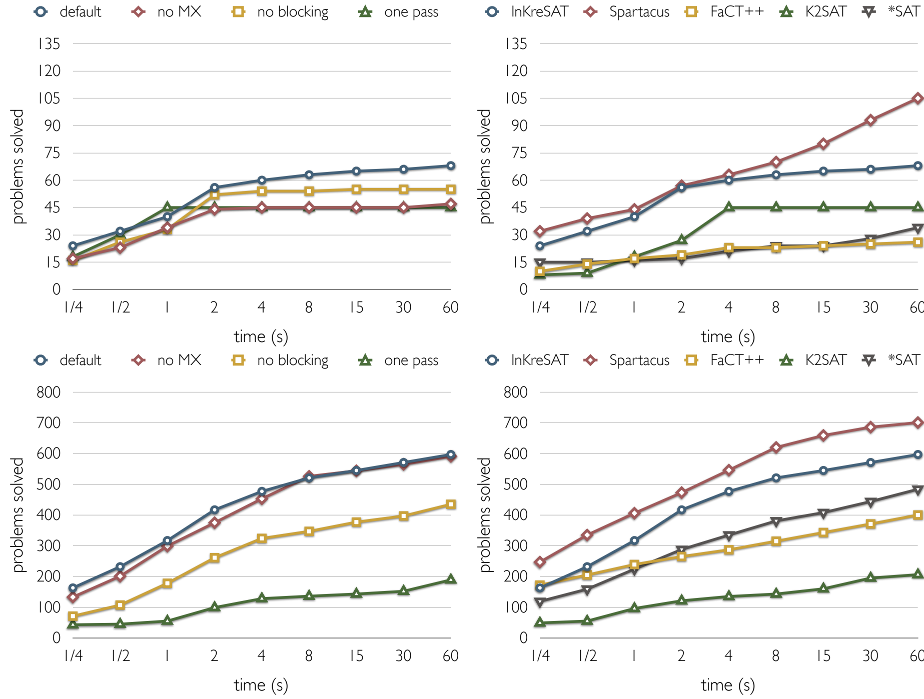

Fig. 2. Results on 3CNF_K (upper half) and MQBF formulas (lower half)

TANCS-2000 [14] Unbounded Modal QBF (MQBF) benchmarks for K complemented by randomly generated modalized MQBF formulas [13] (800 problems in total). In selecting the MQBF problems, we follow [9], but restrict ourselves to the "easy/medium" and "medium" problem classes because of our time limit of 60s. For the same reason, we leave out the harder subclasses of non-modalized "medium" problems (keeping only the problems with V=4, see [9,13,14,10]).

We observe that incremental translation and blocking both lead to considerable performance gains on all benchmarks. With MX, the results are mixed. On LWB, InKreSAT generally performs better without MX. On MQBF, MX makes little difference. On 3CNF_K, however, it is MX that makes blocking efficient and allows InKreSAT to solve more formulas of high modal depth (solving 45/13/10 formulas of depth 2/4/6, resp., compared to 45/9/1 without blocking). Without MX, the overhead caused by blocking actually diminishes performance (to 45/2/0). Compared to the other systems, InKreSAT proves competitive, solving a number of problems that cannot be solved by others, and displaying the arguably best results (without MX) on LWB-K. Note also that in the "one phase" mode, the behavior of InKreSAT expectedly resembles that of K2SAT, K2SAT being slightly faster because of additional optimizations that do not work with incremental translation. A notable weakness of InKreSAT as compared to tableau provers is a faster degradation of performance with increasing modal depth (on LWB-S4 and especially on 3CNF_K, where, e.g., Spartacus solves

40/38/27 problems of depth 2/4/6). We attribute the faster degradation to the higher overhead of blocking in the present setting and to a lack of a more efficient heuristic to guide clause generation. Solving these problems, as well as extending the approach to more expressive logics (e.g., logics with nominals or converse modalities), are interesting directions for future work.

Acknowledgments. This work was partially supported by the EPSRC project Score!.

References

1. Baader, F., Buchheit, M., Hollunder, B.: Cardinality restrictions on concepts. Artif. Intell. 88(1–2), 195–213 (1996)
2. Balsiger, P., Heuerding, A., Schwendimann, S.: A benchmark method for the propositional modal logics K, KT, S4. J. Autom. Reasoning 24(3), 297–317 (2000)
3. Blackburn, P., van Benthem, J., Wolter, F. (eds.): Handbook of Modal Logic. Elsevier (2007)
4. Brown, C.E.: Encoding higher-order theorem proving to a sequence of SAT problems. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE-23. LNCS, vol. 6803, pp. 147–161. Springer (2011)
5. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer (2003)
6. Fitting, M.: Modal proof theory. In: Blackburn et al. [3], pp. 85–138
7. Ganzinger, H., Korovin, K.: New directions in instantiation-based theorem proving. In: LICS 2003. pp. 55–64. IEEE Computer Society Press (2003)
8. Giunchiglia, E., Giunchiglia, F., Tacchella, A.: SAT-based decision procedures for classical modal logics. J. Autom. Reasoning 28(2), 143–171 (2002)
9. Giunchiglia, E., Tacchella, A.: A subset-matching size-bounded cache for testing satisfiability in modal logics. Ann. Math. Artif. Intell. 33(1), 39–67 (2001)
10. Götzmann, D., Kaminski, M., Smolka, G.: Spartacus: A tableau prover for hybrid logic. In: Bolander, T., Braüner, T. (eds.) M4M-6. Electr. Notes Theor. Comput. Sci., vol. 262, pp. 127–139. Elsevier (2010)
11. Horrocks, I., Hustadt, U., Sattler, U., Schmidt, R.: Computational modal logic. In: Blackburn et al. [3], pp. 181–245
12. Kaminski, M., Smolka, G.: Terminating tableau systems for hybrid logic with difference and converse. J. Log. Lang. Inf. 18(4), 437–464 (2009)
13. Massacci, F.: Design and results of the Tableaux-99 non-classical (modal) systems comparison. In: Murray, N.V. (ed.) TABLEAUX '99. LNCS, vol. 1617, pp. 14–18. Springer (1999)
14. Massacci, F., Donini, F.M.: Design and results of TANCS-2000 non-classical (modal) systems comparison. In: Dyckhoff, R. (ed.) TABLEAUX 2000. LNCS, vol. 1847, pp. 52–56. Springer (2000)
15. Sebastiani, R., Vescovi, M.: Automated reasoning in modal and description logics via SAT encoding: The case study of $K_m/\mathcal{ALC}$-satisfiability. J. Artif. Intell. Res. 35, 343–389 (2009)
16. Tacchella, A.: *SAT system description. In: Lambrix, P., Borgida, A., Lenzerini, M., Möller, R., Patel-Schneider, P. (eds.) DL'99. CEUR-WS.org, vol. 22 (1999)
17. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 292–297. Springer (2006)