

# InKreSAT: Modal Reasoning via Reduction to SAT

Mark Kaminski and Tobias Tebbi

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 our approach applicable to the logics K4 and S4.

## 1 Introduction

InKreSAT is a prover for the modal logics K, T, K4, and S4 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 [12] and higher-order logic [4].

To deal with transitivity as it occurs in K4 and S4, as well as to further improve the overall performance of InKreSAT, we extend our basic approach by pattern-based blocking [11].

We evaluate InKreSAT, confirming the effectiveness of our incremental approach compared to the one-phase approach in [15]. Moreover, the performance of InKreSAT proves competitive with that of state-of-the-art modal reasoners based on the tableau method.

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

## 2 Reduction to SAT

We now present the reduction of modal satisfiability to SAT that underlies InKreSAT. We restrict ourselves to the case of multimodal K. An alternative, more detailed presentation of (a variant of) the reduction 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:  $s, t ::= p \mid \neg s \mid s \vee s \mid s \wedge s \mid \langle r \rangle s \mid [r]s$ .

We call formulas of the form  $\langle r \rangle s$  *diamond formulas*, or *diamonds* for short, while formulas  $[r]s$  are called *box formulas* or *boxes*.

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

$$\begin{array}{ccc}
 (\neg) \frac{\sigma : s, \sigma : \sim s}{\otimes} & (\wedge) \frac{\sigma : s_1 \wedge s_2}{\sigma : s_i} \quad i \in \{1, 2\} & (\vee) \frac{\sigma : s_1 \vee s_2}{\sigma : s_1 \mid \sigma : s_2} \\
 (\diamond) \frac{\sigma : \langle r \rangle s}{\tau_{\sigma:\langle r \rangle s} : s} & (\square) \frac{\sigma : [r]s, \sigma : \langle r \rangle t}{\tau_{\sigma:\langle r \rangle t} : s} & 
 \end{array}$$

In the formulation of  $(\neg)$ , we write  $\sim s$  for the negation normal form of  $\neg s$ , while the symbol  $\otimes$  stands for the empty conclusion, representing a closed 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  $s$  of K is satisfiable if and only if there is a maximal tableau rooted at  $\sigma : s$  (for an arbitrary prefix  $\sigma$ ) that has an open branch. For an introduction to tableau systems for modal logic, see [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  $\sigma : s$  to a literal  $l_{\sigma:s}$  such that  $l_{\sigma:s} = \overline{l_{\sigma:\sim s}}$ . Note that all of the above rules have the form  $\frac{\sigma_1:s_1, \dots, \sigma_m:s_m}{\tau_1:t_1 \mid \dots \mid \tau_n:t_n}$  where  $m \in \{1, 2\}$  and  $n \in \{0, \dots, 2\}$ . Thus, we can use the following mapping to assign a clause to every instance of the above rules.

$$\frac{\sigma_1 : s_1, \dots, \sigma_m : s_m}{\tau_1 : t_1 \mid \dots \mid \tau_n : t_n} \rightsquigarrow \overline{l_{\sigma_1:s_1}} \vee \dots \vee \overline{l_{\sigma_m:s_m}} \vee l_{\tau_1:t_1} \vee \dots \vee l_{\tau_n:t_n}$$

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

$$\begin{array}{ccc}
 \sigma : (p \vee q) \wedge \neg p & \rightsquigarrow & l_{\sigma:(p \vee q) \wedge \neg p} \\
 \sigma : p \vee q & (\wedge) \rightsquigarrow & \overline{l_{\sigma:(p \vee q) \wedge \neg p}} \vee l_{\sigma:p \vee q} \\
 \sigma : \neg p & (\wedge) \rightsquigarrow & \overline{l_{\sigma:(p \vee q) \wedge \neg p}} \vee l_{\sigma:\neg p} \\
 \frac{\sigma : p \quad \sigma : q}{\otimes} & (\vee) \rightsquigarrow & \overline{l_{\sigma:p \vee q}} \vee l_{\sigma:p} \vee l_{\sigma:q} \\
 & (\neg) \rightsquigarrow & \overline{l_{\sigma:p}} \vee \overline{l_{\sigma:\neg p}} \quad (\text{redundant})
 \end{array}$$

Note that the prefixed formula  $\sigma : (p \vee q) \wedge \neg p$  at the root of the tableau is mapped to the corresponding unit clause since it is considered an assumption

rather than a consequence of a tableau rule application. Also note that the last clause, which corresponds 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. 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 a SAT solver by adding new clauses to the solver and periodically running the solver on the clauses added so far. In case the solver returns satisfiable, it also returns a satisfying model that is used in selecting new clauses to be added.

Since our mapping of prefixed formulas to literals ensures that literals corresponding to trivially contradictory labeled formulas are contradictory, we never have to generate clauses for the rule  $(\neg)$ . Let  $s$  be a conjunction, disjunction, or a diamond. We write  $C_{\sigma}s$  for the clause corresponding to the instance of  $(\wedge)$ ,  $(\vee)$ , or  $(\diamond)$ , respectively, that has  $\sigma : s$  as its unique premise. We also write  $B_{\sigma}st$ , where  $s$  is a box and  $t$  a diamond, for the clause corresponding to the instance of  $(\square)$  that has  $\sigma : s$  and  $\sigma : t$  as its premises.

The *premise of a clause*  $C$ , where  $C$  corresponds to an instance of a tableau rule  $\frac{\sigma_1:s_1, \dots, \sigma_m:s_m}{\tau_1:t_1 | \dots | \tau_n:t_n}$ , consists of the literals  $l_{\sigma_1:s_1}, \dots, l_{\sigma_m:s_m}$ . We call a labeled formula  $\sigma : s$  *processed* if  $l_{\sigma:s}$  occurs in the premise of some clause previously added to the SAT solver. Otherwise, we call  $\sigma : s$  *pending*, but only if (1)  $\sigma : s$  occurs in some clause added to the SAT solver and (2)  $s$  is not of the form  $p$  or  $\neg p$ . We exclude formulas  $\sigma : p$  and  $\sigma : \neg p$  since they are automatically taken care of by our mapping to literals.

The basic algorithm is shown in Fig. 1. It maintains an agenda consisting of pending labeled formulas that are true in the model returned by a preceding invocation of the SAT solver (or the input formula if the first invocation of the SAT solver is yet to occur). Every formula  $\sigma : s$  on the agenda is processed by adding clauses with  $l_{\sigma:s}$  in their premise, after which  $\sigma : s$  changes its status to processed.

### 4 Blocking

Blocking is a technique commonly employed in tableau calculi to achieve termination in the presence of transitive relations or background theories [10]. Moreover, as shown in [9], blocking can improve the performance of tableau-based decision procedures even in cases it is not required for termination. Blocking typically restricts the applicability of the rule  $(\diamond)$ . An application of  $(\diamond)$  to a given tableau branch is blocked if the mechanism can determine that the successor prefix that would be introduced by the application is subsumed by some prefix that is already on the branch.

```

Input: a formula  $s$ 
Variables:  $\text{Agenda} := \{\sigma_0 : s\}$  (for some arbitrary but fixed prefix  $\sigma_0$ )
while  $\text{Agenda} \neq \emptyset$  do:
  1. for all  $\sigma : s \in \text{Agenda}$  do:
    if  $s$  is a diamond then add  $C_\sigma s \cup \{B_\sigma ts \mid \sigma : t \text{ processed, } t \text{ box}\}$  to SAT solver
    else if  $s$  is a box then add  $\{B_\sigma st \mid \sigma : t \text{ processed, } t \text{ diamond}\}$  to SAT solver
    else add  $C_\sigma s$  to SAT solver
  2. run SAT solver
  3. if SAT solver returns unsat then return unsat
    else  $\text{Agenda} := \{\sigma : s \mid \sigma : s \text{ pending, } l_{\sigma,s} \text{ true in model returned by SAT solver}\}$ 
return sat

```

**Fig. 1.** InKreSAT: basic algorithm

Because of the immediate 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] to blocking is problematic. The main problem is that the approach has no explicit structure corresponding to tableau branches. Known blocking techniques are all designed to work on a single tableau branch at a time. Blocking across branches will typically destroy the correctness of a tableau system.

In our case, the propositional model used to guide the translation in step 3 of the main loop in Fig. 1 approximates a tableau branch. This approximation suffices to argue the correctness of blocking restricted to formulas whose corresponding literals are true in the model. To explore the impact of blocking in our setting, we extend the basic algorithm by an implementation of pattern-based blocking [11] (PBB).

Combining blocking with an incremental translation to SAT turns out to be challenging. Unlike tableau provers, which can incrementally update the data structures needed for blocking, we need to recompute the structures from scratch every time we run the SAT solver. This is due to the fact that, in general, propositional models returned by two runs of a SAT solver may differ in an unpredictable way.

Also, while preserving the correctness of our approach, PBB turns out to be insufficient for termination in the presence of transitivity or background theories. The termination of tableau calculi with PBB relies on the fact that every tableau branch can be obtained from the initial formula by a series of tableau rule applications (see [11]). The approximations of branches returned by a SAT solver do not have this property. They may contain consistent parts that are left over from closed branches but cannot be obtained from the initial formula on the current branch approximation (typically, because PBB would prevent their creation). To regain termination, we extend PBB by a mechanism known from anywhere-blocking [1]. Whenever a prefixed formula  $\sigma : \langle a \rangle s$  (i.e., the corresponding literal) is false in a model or is subject to PBB, we block the entire subtree of prefixes rooted at  $\tau_{\sigma:\langle a \rangle s}$ . The processing of formulas prefixed with a

blocked prefix is delayed for as long as the prefix remains blocked. We call this extension of PBB *subtree blocking*.

## 5 Evaluation and Conclusions

We evaluate the effects of incremental translation to SAT and blocking by comparing the performance of InKreSAT running 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 SB” mode, where PBB is enabled, but subtree blocking is disabled (this makes sense for K where blocking is not necessary for termination), and the default mode, where subtree blocking is enabled. Besides, we include the results from four other 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 command-line options `-j -u -v -w` recommended by the authors. (2) \*SAT [8] (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 DPLL-based SAT solver only for propositional reasoning, while modal reasoning is handled by a conventional tableau calculus. (3) FaCT++ [16] (v1.5.3), an established reasoner for the description logic  $\mathcal{SROIQ}$  with datatypes. (4) Spartacus [9] (v1.1.3), an efficient prover for the hybrid logic  $\mathcal{H}(E, @)$ . While K2SAT and \*SAT are included because they implement related approaches, 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 (in contrast to an evaluation in [7], where \*SAT is compiled with non-default settings).

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 used in [9]). **Table 1:** The K and S4 problem sets from the Logic Work Bench (LWB) benchmarks [2]. For each subclass that was not solved in its hardest instance (21) by every system, Table 1 displays the hardest problem 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 transitive relations. **Fig. 2, upper half:** Randomly generated  $3CNF_K$  formulas [7] of modal depth 2, 4, and 6 (45 problems each, 135 in total; see [9, 7]). 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 TANCS-2000 Unbounded Modal QBF (MQBF) benchmarks for K [14] complemented by randomly generated modalized MQBF formulas [13] (800 problems in total; see [9]).

We observe that, when run in the “one phase” mode, the behaviour of InKreSAT generally resembles that of K2SAT, K2SAT performing slightly better because of some additional optimizations. The addition of incremental transla-

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

Subclass	InKreSAT (default)	Spartacus	FaCT++
branch_n	<b>11</b>	9	6
md_n	8	<b>21</b>	10
md_p	3	<b>9</b>	4
ipc_n	9	<b>21</b>	10
ipc_p	8	<b>21</b>	9
path_n	4	16	<b>21</b>
path_p	5	17	<b>21</b>
ph_n	<b>13</b>	10	8
ph_p	<b>9</b>	5	6
s5_n	14	16	<b>19</b>

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

tion, PBB, and subset blocking all lead to considerable improvements in performance on all benchmarks. Thus, with all of the techniques in place, InKreSAT displays a competitive performance (in fact, solving a number of problems that cannot be solved by other systems). One notable weakness of InKreSAT as compared to tableau provers is a somewhat faster degradation of performance with increasing modal depth (which can be seen on the S4 and  $3\text{CNF}_K$  benchmarks). We attribute this to the higher overhead of blocking in the present setting (see §4) and to a lack of a more efficient heuristic to guide the clause generation. Solving these problems, as well as extending the approach to more expressive logics, are interesting directions for future work.

## References

1. Baader, F., Buchheit, M., Hollunder, B.: Cardinality restrictions on concepts. *Artif. Intell.* 88(1–2), 195–213 (1996)
2. Balsiger, P., Heurding, 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, Studies in Logic and Practical Reasoning*, vol. 3. 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. Giunchiglia, E., Giunchiglia, F., Tacchella, A.: SAT-based decision procedures for classical modal logics. *J. Autom. Reasoning* 28(2), 143–171 (2002)
8. Giunchiglia, E., Tacchella, A.: System description: \*SAT: A platform for the development of modal decision procedures. In: McAllester, D.A. (ed.) *CADE-17. LNCS*, vol. 1831, pp. 291–296. Springer (2000)

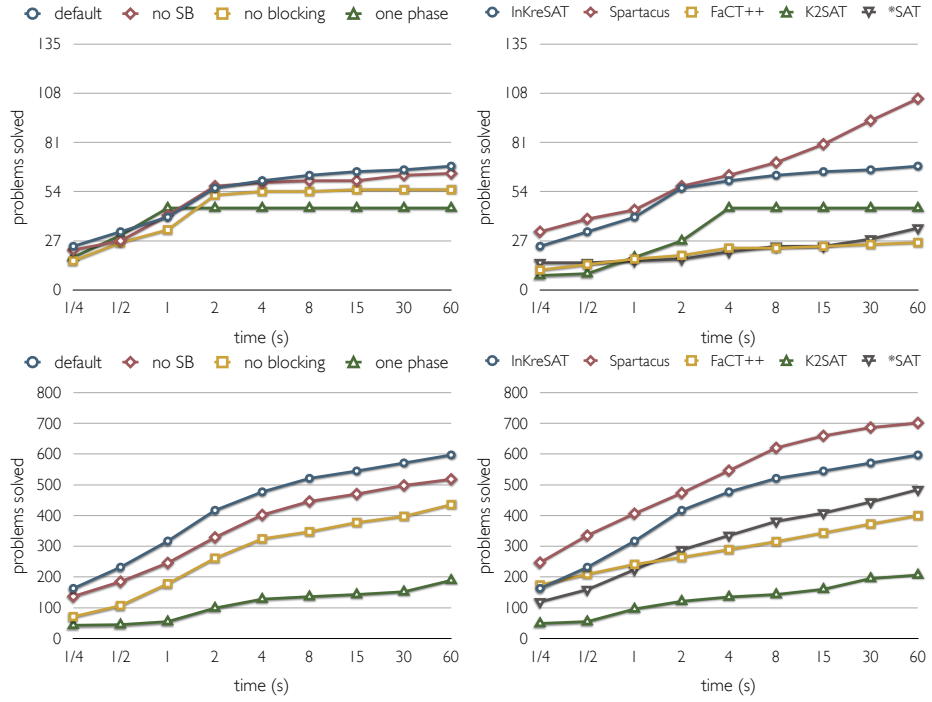


Fig. 2. Results on 3CNF<sub>K</sub> (upper half) and MQBF formulas (lower half)

9. 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)
10. Horrocks, I., Hustadt, U., Sattler, U., Schmidt, R.: Computational modal logic. In: Blackburn et al. [3], pp. 181–245
11. Kaminski, M., Smolka, G.: Terminating tableau systems for hybrid logic with difference and converse. J. Log. Lang. Inf. 18(4), 437–464 (2009)
12. Korovin, K.: iProver – an instantiation-based theorem prover for first-order logic (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 292–298. Springer (2008)
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. 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)