$\mathsf{K_SP}$: A resolution-based prover for multimodal K

Cláudia Nalon[1], Ullrich Hustadt[2], and Clare Dixon[2]

[1] Department of Computer Science, University of Brasília
C.P. 4466 – CEP:70.910-090 – Brasília – DF – Brazil
nalon@unb.br

[2] Department of Computer Science, University of Liverpool
Liverpool, L69 3BX – United Kingdom
U.Hustadt, CLDixon@liverpool.ac.uk

Abstract. In this paper, we describe an implementation of a hyper-resolution-based calculus for the propositional basic multimodal logic, K_n. The prover was designed to support experimentation with different combinations of refinements for its basic calculus: it is primarily based on the set of support strategy, which can then be combined with other refinements, simplification techniques and different choices for the underlying normal form and clause selection. The prover allows for both local and global reasoning. We show experimental results for different combinations of strategies and comparison with existing tools.

1 Introduction

In this paper, we present $\mathsf{K_SP}$, a theorem prover for the basic multimodal logic K_n which implements a variation of the set of support strategy [20] for the modal resolution-based procedure described in [13]. The prover also implements several other refinements and simplification techniques in order to reduce the search space for a proof. Besides the set of support strategy, all other refinements of the calculus are implemented as independent modules, allowing for a better evaluation of how effective they are.

The paper is organised as follows. We introduce the syntax and semantics of K_n in Section 2. In Section 3 we briefly describe the normal form and the calculus presented in [13]. Section 4 describes the available strategies and their implementations. Evaluation of strategies and of the performance of the prover compared to existing tools are given in Section 5. We summarise our results in Section 6.

2 Language

Let $\mathsf{A} = \{1, \ldots, n\}$, $n \in \mathbb{N}$, be a finite fixed set of indexes and $\mathsf{P} = \{p, q, s, t, p', q', \ldots\}$ be a denumerable set of propositional symbols. The *set of modal formulae*, $\mathsf{WFF_K}$, is the least set such that every $p \in \mathsf{P}$ is in $\mathsf{WFF_K}$; if φ and ψ are in $\mathsf{WFF_K}$, then so are $\neg\varphi$, $(\varphi \wedge \psi)$, and $\boxed{a}\varphi$ for each $a \in \mathsf{A}$. The formulae **false**, **true**, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$, and $\Diamond_{a}\varphi$ are introduced as the usual abbreviations for $(\varphi \wedge \neg\varphi)$, $\neg$**false**, $\neg(\neg\varphi \wedge \neg\psi)$, $(\neg\varphi \vee \psi)$, and $\neg\boxed{a}\neg\varphi$, respectively (where $\varphi, \psi \in \mathsf{WFF_K}$). A *literal* is either a propositional symbol or its negation; the set of literals is denoted by L. A *modal* *literal* is either $\boxed{a}l$ or $\Diamond_a l$, where $l \in \mathsf{L}$ and $a \in \mathsf{A}$. The *modal depth* of a formula is given by the maximal number of nested occurrences of modal operators in that formula. The *modal level* of a formula is the maximal number of nested occurrences of modal operators in which scope the formula occurs. For instance, in $\boxed{a}\Diamond_a p$, the modal depth of p is 0 and its modal level is 2. Formal definitions can be found at [13].

As our calculus operates on a labelled clausal normal form that is closely linked to the tree model property of Kripke models for K_n, we briefly overview of the semantics of K. A *tree-like Kripke model M for n agents over* P is given by a tuple $(W, w_0, R_1, \ldots, R_n, \pi)$, where W is a set of possible *worlds* with a distinguished world w_0, each *accessibility relation R_a* is a binary relation on W such that their union is a tree with *root* w_0, and $\pi : W \to (\mathsf{P} \to \{\mathit{true}, \mathit{false}\})$ is a function which associates with each world $w \in W$ a truth-assignment to propositional symbols. Satisfaction of a formula at a world w of a model M is defined by:

- $\langle M, w\rangle \models p$ if, and only if, $\pi(w)(p) = \mathit{true}$, where $p \in \mathsf{P}$;
- $\langle M, w\rangle \models \neg\varphi$ if, and only if, $\langle M, w\rangle \not\models \varphi$;
- $\langle M, w\rangle \models (\varphi \wedge \psi)$ if, and only if, $\langle M, w\rangle \models \varphi$ and $\langle M, w\rangle \models \psi$;
- $\langle M, w\rangle \models \boxed{a}\varphi$ if, and only if, for all w', wR_aw' implies $\langle M, w'\rangle \models \varphi$.

Let $M = (W, w_0, R_1, \ldots, R_n, \pi)$ be a model. A formula φ is *locally satisfied in M*, denoted by $M \models_L \varphi$, if $\langle M, w_0\rangle \models \varphi$. The formula φ is *locally satisfiable* if there is a model M such that $\langle M, w_0\rangle \models \varphi$. A formula φ is *globally satisfied in M*, if for all $w \in W$, $\langle M, w\rangle \models \varphi$. We denote by $\mathsf{depth}(w)$ the length of the unique path from w_0 to w through the union of the accessibility relations in M. We call a *modal layer* the equivalence class of worlds at the same depth in a model.

We note that checking the local satisfiability of a formula φ can be reduced to the problem of checking the local satisfiability of its subformulae at the modal layer of a model which corresponds to the modal level where those subformulae occur (see [1]). Due to this close correspondence of modal layer and modal level we use the terms interchangeably. Also, checking the global satisfiability of φ can be reduced to checking the local satisfiability of φ at all modal layers (up to an exponential distance from the root) of a model [3,18]. Thus, an uniform approach based on modal levels can be used to deal with both problems, as we show in the next section.

3 A Calculus for K_n

The calculus for K_n presented in [13] is clausal, where clauses are labelled by the modal level at which they occur. In order to refer explicitly to modal levels, the modal language is extended with labels. We write $ml : \varphi$ to denote that φ is true at the modal layer ml in a Kripke model, where $ml \in \mathbb{N} \cup \{*\}$. By $* : \varphi$ we mean that φ is true at all modal layers in a Kripke model. The notion of local satisfiability is extended as expected: for a model M, $M \models_L ml : \varphi$ if, and only if, for all worlds $w \in W$ such that $\mathsf{depth}(w) = ml$, we have $\langle M, w\rangle \models_L \varphi$. Then, the layered normal form, called SNF_{ml}, is given by a conjunction of *literal clauses* of the form $ml : D$, where D is a disjunction of literals, and *modal clauses* of the form $ml : l \Rightarrow l'$, where $l \in \mathsf{L}$ and l' is a modal literal. Transformation into SNF_{ml} uses *renaming* and preserves satisfiability.

The motivation for the use of this labelled clausal normal form is that inference rules can then be guided by the semantic information given by the labels and applied to

[LRES]
$$\begin{array}{r}
ml : D \lor l \\
ml' : D' \lor \neg l \\
\hline
ml : D \lor D'
\end{array}$$

[MRES]
$$\begin{array}{r}
ml : l_1 \Rightarrow \boxed{a} l \\
ml' : l_2 \Rightarrow \langle a \rangle \neg l \\
\hline
ml : \neg l_1 \lor \neg l_2
\end{array}$$

[GEN2]
$$\begin{array}{r}
ml_1 : l'_1 \Rightarrow \boxed{a} l_1 \\
ml_2 : l'_2 \Rightarrow \boxed{a} \neg l_1 \\
ml_3 : l'_3 \Rightarrow \langle a \rangle l_2 \\
\hline
ml : \neg l'_1 \lor \neg l'_2 \lor \neg l'_3
\end{array}$$

[GEN1]
$$\begin{array}{r}
ml_1 : l'_1 \Rightarrow \boxed{a} \neg l_1 \\
\vdots \\
ml_m : l'_m \Rightarrow \boxed{a} \neg l_m \\
ml_{m+1} : l' \Rightarrow \langle a \rangle \neg l \\
ml_{m+2} : l_1 \lor \ldots \lor l_m \lor l \\
\hline
ml : \neg l'_1 \lor \ldots \lor \neg l'_m \lor \neg l'
\end{array}$$

[GEN3]
$$\begin{array}{r}
ml_1 : l'_1 \Rightarrow \boxed{a} \neg l_1 \\
\vdots \\
ml_m : l'_m \Rightarrow \boxed{a} \neg l_m \\
ml_{m+1} : l' \Rightarrow \langle a \rangle l \\
ml_{m+2} : l_1 \lor \ldots \lor l_m \\
\hline
ml : \neg l'_1 \lor \ldots \lor \neg l'_m \lor \neg l'
\end{array}$$

Table 1: Inference rules, where $ml = \sigma(\{ml_1, \ldots, ml_{m+1}, ml_{m+2} - 1\})$ in GEN1, GEN3; $ml = \sigma(\{ml, ml'\})$ in LRES, MRES; and $ml = \sigma(\{ml_1, ml_2, ml_3\})$ in GEN2.

smaller sets of clauses, reducing the number of unnecessary inferences, and therefore improving the efficiency of the proof procedure. The calculus comprises the set of inference rules given in Table 1. Unification on sets of labels is defined by $\sigma(\{ml, *\}) = ml$; and $\sigma(\{ml\}) = ml$; otherwise, σ is undefined. The inference rules can only be applied if the unification of their labels is defined (where $* - 1 = *$). This calculus has been shown to be sound, complete, and terminating [13].

4 Implementation

KSP is an implementation, written in C, of the calculus given in [13]. The prover was designed to support experimentation with different combinations of refinements of its basic calculus. Refinements and options for (pre)processing the input are coded as independently as possible in order to allow for the easy addition and testing of new features. This might not lead to optimal performance (e.g. one technique needs to be applied after the other, whereas most tools would apply them together), but it helps to evaluate how the different options independently contribute to achieve efficiency. In the following we give a brief overview of the main aspects of the implementation.

Transformation to clausal form: If the input is a set of formulae, then these formulae are first transformed into their prenex or antiprenex normal form (or one after the other) [11] and then into Negation Normal Form (NNF) or into Box Normal Form (BNF) [15]. With options *nnfsimp* (resp. *bnfsimp*), simplification is applied to formulae in NNF (resp. BNF); with option *early_mlple*, pure literal elimination is applied at every modal level. There are then four different options that determine the normal form. In SNF^{+}_{ml}, negative literals in the scope of modal operators are renamed by propositional symbols; in SNF^{++}_{ml} all literals in the scope of modal operators are renamed by propositional symbols. SNF^{-}_{ml} and SNF^{--}_{ml} are defined analogously, with positive literals being renamed by negative ones. The reuse of propositional symbols in renaming can also be controlled. In our evaluation, given in Section 5, the same propositional symbol is used for all occurrences of a formula being renamed.

Preprocessing of clauses: Self-subsumption is applied at this step if the options for forward and/or backward subsumption are set [**?**]. The inference rules MRES and GEN2 are also exhaustively applied at this step, that is, before the prover enters the main loop.

Main loop: The main loop is based on the given-clause algorithm implemented in Otter [10], a variation of the set of support strategy [20], a refinement which restricts the set of choices of clauses participating in a derivation step. For the classical case, a set of clauses Δ is partitioned in two sets Γ and $\Lambda = \Delta \setminus \Gamma$, where Λ must be satisfiable. McCune refers to Γ as the *set of support* (the sos, aka *passive* or *unprocessed* set); and Λ is called the *usable* (aka as *active* or *processed* set). The *given clause* is chosen from Γ, resolved with clauses in Λ, and moved from Γ to Λ. Resolvents are added to Γ. For the modal calculus, the set of clauses is further partitioned according to the modal layer at which clauses are true. That is, for each modal layer ml there are three sets: Γ^{lit}_{ml}, Λ^{lit}_{ml} and Λ^{mod}_{ml}, where the first two sets contain literal clauses while the latter contains modal clauses. As the calculus does not generate new modal clauses and because the set of modal clauses by itself is satisfiable, there is no need for a set for unprocessed modal clauses. Attempts to apply an inference rule are guided by the choice, for each modal layer ml, of a literal clause in Γ^{lit}_{ml}, which can be resolved with either a literal clause in Λ^{lit}_{ml} or with a set of modal clauses in Λ^{mod}_{ml-1}. There are six options for automatically populating the usable: all negative clauses, all positive clauses, all non-negative clauses, all non-positive clauses, all clauses whose maximal literal is positive, and all clauses whose maximal literal is negative. The prover can either perform *local* or *global* reasoning.

Refinements: Besides the basic calculus with a set-of-support strategy, the user can further restrict LRES by choosing ordered (clauses can only be resolved on their maximal literals with respect to an ordering chosen by the prover in such a way to preserve completeness), negative (one of the premises is a negative clause, i.e. a clause where all literals are of the form $\neg p$ for some $p \in \mathsf{P}$), positive (one of the premises is a positive clause), or negative + ordered resolution (both negative and ordered resolution inferences are performed).

The completeness of some of these refinements depends on the particular normal form chosen. For instance, negative resolution is incomplete without SNF^{+}_{ml} or SNF^{++}_{ml}. For example, the set $\{p, p \Rightarrow \Box\neg q, p \Rightarrow \Diamond s, \neg s \vee q\}$ is unsatisfiable, but as there is no negative literal clause in the set, no refutation can be found. By renaming $\neg q$ in the scope of $\Box$, we obtain the set $\{p, p \Rightarrow \Box t, p \Rightarrow \Diamond s, \neg s \vee q, \neg t \vee \neg q\}$ in SNF^{+}_{ml}, from which a refutation using negative resolution can be found. Similarly, ordered resolution requires SNF^{++}_{ml} for completeness, while positive resolution requires SNF^{-}_{ml} or SNF^{--}_{ml}.

Inference rules: Besides the inference rules given in Table 1, three more inference rules are also implemented: *unit resolution*, which propagates unit clauses through all literal clauses and the right-hand side of modal clauses; *lhs unit resolution*, which propagates unit clauses through the left-hand side of modal clauses; and *ires*, which together with the *global* option, implements initial resolution and, thus, the calculus given in [12].

Redundancy elimination: Pure literal elimination can be applied globally or by modal layer. Both forward and backward subsumption are implemented. Subsumption is applied in lazy mode: a clause is tested for subsumption only when it is selected from the sos and only against clauses in the usable. As pointed out in [17], this avoids expensive checks for clauses that might never be selected during the search of a proof.

Clause selection: There are five different heuristics for choosing a literal clause in the sos: shortest, newest, oldest, greatest maximal literal, and the smallest maximal literal.

For a comprehensive list of options, see [14], where the sources and instructions on how to install and use K_SP can be found.

5 Evaluation

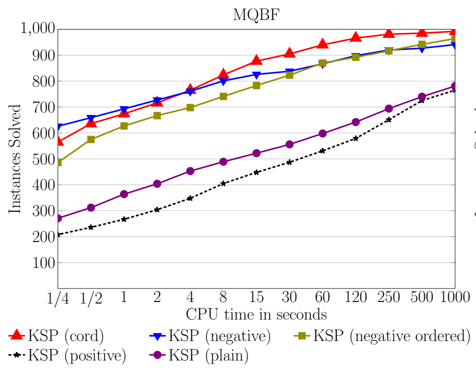
We have compared K_SP with BDDTab [4], FaCT++ 1.6.3 [19], InKreSAT 1.0 [7], Spartacus 1.0 [5], and a combination of the optimised functional translation [6] with Vampire 3.0 [8] [3]. In this context, FaCT++ represents the previous generation of reasoners while the remaining systems have all been developed in recent years. Unless stated otherwise, the reasoners were used with their default options.

Our benchmarks [14] consist of three collections of modal formulae:

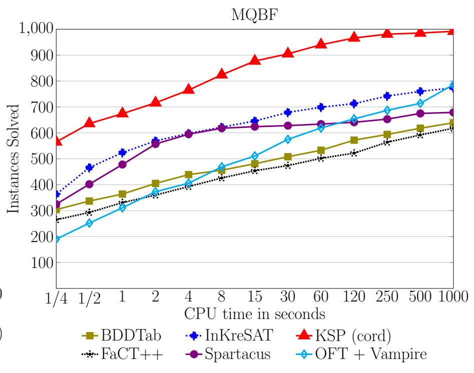
1. The complete set of TANCS-2000 modalised random QBF (MQBF) formulae [9] complemented by the additional MQBF formulae provided by Tebbi and Kaminski [7]. This collection consists of five classes, called qbf, qbfL, qbfS, qbfML, and qbfMS in the following, with a total of 1016 formulae, of which 617 are known to be satisfiable and 399 are known to be unsatisfiable (due to at least one of the provers being able to solve the formula). The minimum modal depth of formulae in this collection is 19, the maximum 225, average 69.2 with a standard deviation of 47.5.
2. LWB basic modal logic benchmark formulae [2], with 56 formulae chosen from each of the 18 parameterised classes. In most previous uses of these benchmarks, only parameter values 1 to 21 were used for each class, with the result that provers were able to solve all benchmark formulae for most of the classes. Instead we have chosen the 56 parameter values so that the best current prover will not be able to solve all the formulae within a time limit of 1000 CPU second. The median value of the maximal parameter value used for the 18 classes is 1880, far beyond what has ever been tested before. Of the resulting 1008 formulae, half are satisfiable and half are unsatisfiable by construction of the benchmark classes.
3. Randomly generated $3CNF_K$ formulae [16] over 3 to 10 propositional symbols with modal depth 1 or 2. We have chosen formulae from each of the 11 parameter settings given in the table on page 372 of [16]. For the number of conjuncts we have focused on a range around the critical region where about half of the generated formulae are satisfiable and half are unsatisfiable. The resulting collection contains 1000 formulae, of which 457 are known to be satisfiable and 464 are known to be unsatisfiable. Note that this collection is quite distinct to the one used in [7] which consisted of 135 $3CNF_K$ formulae over 3 propositional symbols with modal depth 2, 4 or 6, all of which were satisfiable.

Benchmarking was performed on PCs with an Intel i7-2600 CPU @ 3.40GHz and 16GB main memory. For each formula and each prover we have determined the median run time over five runs with a time limit of 1000 CPU seconds for each run.

[3] We have excluded *SAT from the comparison as it produced incorrect results on a number of benchmark formulae.



(a) K$_\mathsf{S}$P with different refinements



(b) All provers

Fig. 1: Benchmarking results for MQBF

Figure 1a compares the impact of different refinements on the performance of K$_\mathsf{S}$P on the MQBF collection. With *plain* K$_\mathsf{S}$P uses the rules shown in Table 1, without additional refinement, on a set of SNF_{ml}^{++} clauses. With *cord* and *negative ordered*, K$_\mathsf{S}$P applies ordered resolution and negative + ordered resolution, respectively, again on a set of SNF_{ml}^{++} clauses. The configuration *negative* uses negative resolution on a set of SNF_{ml}^{+} clauses, while with *positive*, K$_\mathsf{S}$P applies positive resolution on SNF_{ml}^{-} clauses. Irrespective of the refinement, the shortest clause is selected to perform inferences; both forward and backward subsumption are used; the lhs-unit resolution rule is applied; prenex is set; and no simplification steps are applied. The usable is populated with clauses whose maximal literal is positive, except for positive resolution where it is populated with clauses whose maximal literal is negative. The *cord* configuration offers the best performance. Ordered resolution restricts the applicability of the rules further than the other refinements. Not only is this an advantage on satisfiable formulae in that a saturation can be found more quickly, but also unsatisfiable formulae where with this refinement K$_\mathsf{S}$P finds refutations much more quickly than with any of the other refinements.

Figure 1b compares the performance of all the provers on the MQBF collection. It shows that K$_\mathsf{S}$P performs better than any of the other provers. The graphs in Figure 2 offer some insight into why K$_\mathsf{S}$P performs well on these formulae. Each of the five graphs shows for one formula from each class how many atomic subformulae occur at

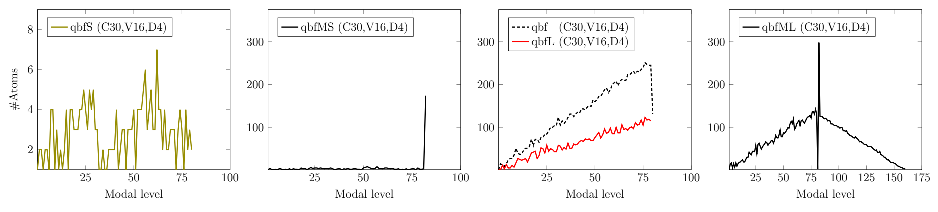


Fig. 2: Modal structure of MQBF formulae

	BDDTab		FaCT++		InKreSAT		K$_{\mathsf{S}}$P (cord)		Spartacus		OFT + Vampire	
k_branch_n	22	*22*	12	*12*	15	*15*	18	*18*	12	*12*	**50**	*70*
k_branch_p	22	*22*	12	*12*	22	*22*	23	*23*	14	*14*	**50**	*70*
k_d4_n	20	*440*	6	*40*	34		**48**	*1560*	28	*760*	14	*200*
k_d4_p	26	*640*	24	*600*	18	*360*	**54**	*1800*	32	*920*	21	*960*
k_dum_n	39	*2400*	42	*2640*	23	*1120*	**49**	*3200*	44	*2800*	17	*640*
k_dum_p	42	*2640*	38	*2320*	28	*1520*	**50**	*3280*	46	*2960*	18	*720*
k_grz_n	35	*2600*	27	*1800*	50	*4500*	5	*50*	**52**	*5500*	24	*1500*
k_grz_p	35	*2600*	27	*1800*	51	*5000*	29	*2000*	**52**	*5500*	27	*1800*
k_lin_n	46	*4000*	43	*3400*	33	*2500*	1	*10*	**50**	*4800*	40	*3100*
k_lin_p	14	*500*	28	*10000*	**56**	*500000*	23	*5000*	55	*400000*	28	*10000*
k_path_n	37	*290*	48	*400*	7	*14*	**54**	*1000*	48	*400*	41	*330*
k_path_p	35	*270*	48	*400*	5	*12*	**54**	*1000*	48	*400*	41	*330*
k_ph_n	10	*10*	8	*16*	24	*90*	3	*6*	**21**	*75*	15	*45*
k_ph_p	**11**	*11*	9	*8*	10	*10*	5	*5*	9	*9*	10	*10*
k_poly_n	39	*600*	34	*500*	30		36	*540*	**44**	*720*	20	*220*
k_poly_p	38	*580*	34	*500*	28	*400*	36	*540*	**44**	*700*	20	*220*
k_t4p_n	40	*3500*	24	*1500*	17	*800*	39	*3000*	**45**	*6000*	11	*200*
k_t4p_p	48	*7500*	49	*8000*	28		49	*8000*	**53**	*12000*	14	*500*

Table 2: Detailed benchmarking results on LWB

each modal level, the formulae originate from MQBF formulae with the same number of propositional symbols, conjuncts and QBF quantifier depth. Formulae in the class qbfS are the easiest, the total number of atomic subformulae is low and spread over a wide range of modal levels, thereby reducing the possibility of inference steps between the clauses in the layered normal form of these formulae. In contrast, in qbfMS formulae almost all atomic subformulae occur at just one modal level. Here the layered normal form can offer little advantage over a simpler normal form. But the number of atomic subformulae is still low and K$_{\mathsf{S}}$P seems to derive an advantage from the fact that the normal form 'flattens' the formula: K$_{\mathsf{S}}$P is at least two orders of magnitude faster than any other prover on this class. The classes qbf and qbfL are more challenging. While the atomic subformulae are more spread out over the modal levels than for qbfMS, at a lot of these modal levels there are more atomic subformulae than in a qbfMS formula in total. The layered modal translation is effective at reducing the number of inferences for these classes, but more inference possibilities remain than for qbfMS. Finally, qbfML combines the worst aspects of qbfL and qbfMS, the number of atomic subformulae is higher than for any other class and there is a 'peak' at one particular modal level. This is the only MQBF class containing formulae that K$_{\mathsf{S}}$P cannot solve.

Figure 3 shows the benchmarking results on the LWB and 3CNF$_K$ collections. On the LWB collection K$_{\mathsf{S}}$P performs about as well as BDDTab, FaCT++ and InKreSAT, while Spartacus performs best and the combination of the optimised functional translation with Vampire (OFT + Vampire) performs worst. Table 2 provides more detailed results. For each prover it shows in the left column how many of the 56 formulae in a class have been solved and in the right column the parameter value of the most difficult formula solved. For InKreSAT we are not reporting this parameter value for three classes on which the prover's runtime does not increase monotonically with the param-

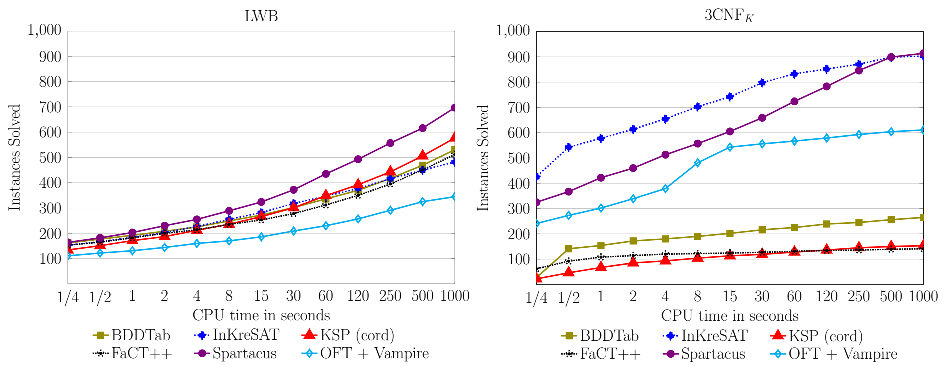


Fig. 3: Benchmarking results for LWB and 3CNF$_K$

eter value but fluctuates instead. As is indicated, BDDTab and InKreSAT are the best performing provers on one class each, OFT + Vampire on two, K$_S$P on six, and Spartacus on eight classes. A characteristic of the classes on which K$_S$P performs best is again that atomic subformulae are evenly spread over a wide range of modal levels.

It is worth pointing out that simplification alone is sufficient to detect that formulae in k_lin_p are unsatisfiable. For k_grz_p, pure literal elimination can be used to reduce all formulae in this class to the same simple formula; the same is true for k_grz_n and k_lin_n. Thus, these classes are tests how effectively and efficiently, if at all, a prover uses these techniques and Spartacus does best on these classes. Note that pure literal elimination has been disabled in the *cord* configuration we have used for K$_S$P. With it K$_S$P would perform better on k_grz_p and k_grz_n, but worse on other classes where this simplification has no beneficial effects.

Finally, on the 3CNF$_K$ collection, InKreSAT is the best performing prover and K$_S$P the worst performing one. This should now not come as a surprise. For 3CNF$_K$ we specifically restricted ourselves to formulae with low modal depth which in turn means that the layered normal form has little positive effect.

6 Conclusions and Future Work

The evaluation indicates that K$_S$P works well on problems with high modal depth where the separation of modal layers can be exploited to improve the efficiency of reasoning.

As with all provers that provide a variety of strategies and optimisations, to get the best performance for a particular formula or class of formulae it is important to choose the right strategy and optimisations. K$_S$P currently leaves that choice to the user and the development of an “auto mode” in which the prover makes a choice of its own, based on an analysis of the given formula, is future work.

The same applies to the transformation to the layered normal form. Again, K$_S$P offers a number of ways in which this can be done as well as a number of simplifications that can be applied during the process. It is clear that this affects the performance of the prover, but we have yet to investigate the effects on the benchmark collections introduced in this paper.

References

1. C. Areces, R. Gennari, J. Heguiabehere, and M. D. Rijke. Tree-based heuristics in modal theorem proving. In *Proc. ECAI 2000*, pages 199–203. IOS Press, 2000.
2. P. Balsiger, A. Heuerding, and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. *J. Autom. Reasoning*, 24(3):297–317, 2000.
3. V. Goranko and S. Passy. Using the universal modality: gains and questions. *Journal of Logic and Computation*, 2(1):5–30, 1992.
4. R. Goré, K. Olesen, and J. Thomson. Implementing tableau calculi using BDDs: BDDTab system description. In *Proc. IJCAR 2014*, volume 8562 of *LNCS*, pages 337–343. Springer, 2014.
5. D. Götzmann, M. Kaminski, and G. Smolka. Spartacus: A tableau prover for hybrid logic. *Electr. Notes Theor. Comput. Sci.*, 262:127–139, 2010.
6. I. R. Horrocks, U. Hustadt, U. Sattler, and R. Schmidt. Computational modal logic. In *Handbook of Modal Logic*, pages 181–245. Elsevier, 2006.
7. M. Kaminski and T. Tebbi. InKreSAT: Modal reasoning via incremental reduction to SAT. In *Proc. CADE-24*, volume 7898 of *LNAI*, pages 436–442. Springer, 2013.
8. L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In *Proc. CAV2013*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
9. F. Massacci and F. M. Donini. Design and results of TANCS-2000 non-classical (modal) systems comparison. In *Proc. TABLEAUX 2000*, volume 1847 of *LNCS*, pages 52–56. Springer, 2000.
10. W. W. McCune. OTTER 3.0 reference manual and guide, May 07 2007.
11. C. Nalon and C. Dixon. Anti-prenexing and prenexing for modal logics. In *Proc. JELIA 2006*, volume 4160 of *LNCS*, pages 333–345. Springer, 2006.
12. C. Nalon and C. Dixon. Clausal resolution for normal modal logics. *J. Algorithms*, 62:117–134, 2007.
13. C. Nalon, U. Hustadt, and C. Dixon. A modal-layered resolution calculus for K. In *Proc. TABLEAUX 2015*, volume 9323 of *LNCS*, pages 185–200. Springer, 2015.
14. C. Nalon, U. Hustadt, and C. Dixon. KSP: sources and benchmarks. `http://www.cic.unb.br/~nalon/#software`, 2016.
15. G. Pan, U. Sattler, and M. Y. Vardi. BDD-based decision procedures for the modal logic K. *Journal of Applied Non-Classical Logics*, 16(1-2):169–208, 2006.
16. P. F. Patel-Schneider and R. Sebastiani. A new general method to generate random modal formulae for testing decision procedures. *J. Artif. Intell. Res. (JAIR)*, 18:351–389, 2003.
17. S. Schulz. Simple and efficient clause subsumption with feature vector indexing. In M. P. Bonacina and M. E. Stickel, editors, *Automated Reasoning and Mathematics*, volume 7788 of *LNCS*, pages 45–67. Springer, 2013.
18. E. Spaan. *Complexity of Modal Logics*. PhD thesis, University of Amsterdam, 1993.
19. D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. IJCAR 2006*, volume 4130 of *LNCS*, pages 292–297. Springer, 2006.
20. L. Wos, G. Robinson, and D. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *Journal of the ACM*, 12:536–541, 1965.