Spartacus: A Tableau Prover for Hybrid Logic

Daniel Götzmann, Mark Kaminski, and Gert Smolka

Saarland University, Saarbrücken, Germany

Abstract. Spartacus is a tableau prover for hybrid multimodal logic with global modalities and reflexive and transitive relations. Spartacus is the first system to use pattern-based blocking for termination. To achieve a competitive performance, Spartacus implements a number of optimization techniques, including a new technique that we call lazy branching. We evaluate the practical impact of pattern-based blocking and lazy branching for the basic modal logic **K** and observe high effectiveness of both techniques.

1 Introduction

Automated reasoning in modal and description logics (DL) is an active field of research. Arguably the most successful approach to modal reasoning are tableau-based methods. Several of the most prominent DL reasoners, including FaCT++ [1] and RacerPro [2], are based on tableau algorithms. In the presence of global modalities or transitive relations, the naive tableau construction strategy, sufficient in the case of basic modal logic, no longer terminates. To regain termination, one employs blocking [3]. Most of the established blocking techniques are derived from Kripke's chain-based approach [4]. Kaminski and Smolka [5, 3] propose a different blocking technique, called pattern-based blocking. They conjecture that pattern-based blocking may display a better performance than the established techniques. Our goal is to show that pattern-based blocking is useful even for **K**, where blocking is not required for termination.

Spartacus is a tableau prover for hybrid multimodal logic with global modalities. It supports reasoning in the presence of reflexive and/or transitive relations. In contrast to other systems, Spartacus uses pattern-based blocking to achieve termination. Crucial for the performance of pattern-based blocking is the data structure used to store blocked patterns (pattern store). Spartacus allows us to evaluate two different implementations of the pattern store, one of them based on a data structure by Giunchiglia and Tacchella [6], the other one on work by Hoffmann and Koehler [7]. Similarly to FaCT++ [1], Spartacus schedules pending rule applications using a configurable priority queue, which allows for a fine-grained control over the rule application strategy. To achieve a reasonable performance on realistic inputs, Spartacus implements a number of optimizations, including term normalization, Boolean constraint propagation, semantic branching and backjumping [8]. Moreover, Spartacus implements a new technique, called *lazy branching*. Lazy branching is a generalization of lazy unfolding [8], an effective optimization technique from DL reasoning. Restricted to propositional reasoning, lazy branching corresponds to the Pure Literal Rule. Spartacus is written in Standard ML and compiled with MLton. The source code and test data are available from `www.ps.uni-sb.de/theses/goetzmann/`. A detailed description of Spartacus can be found in [9].

We evaluate the effects of pattern-based blocking and lazy branching, and compare the performance of Spartacus with that of other reasoners for modal and description logics. Both techniques prove highly effective.

2 Pattern-Based Blocking

Pattern-based blocking (PBB) in Spartacus is implemented following [5]. The *pattern* $P(s)$ of a diamond formula s is a set of formulas consisting of s itself and all the boxes located at the same node as s on the tableau branch (nodes are also known as nominals or prefixes). Once the diamond rule is applied to s, $P(s)$ is marked as *expanded*. Moreover, a pattern P is considered *expanded* if there is an expanded pattern Q such that $P \subseteq Q$. PBB restricts the applicability of the diamond rule to formulas whose patterns are not yet expanded on the branch.

Crucial for the performance of PBB is the ability to efficiently determine, given a query pattern P, whether there is an expanded superpattern (i.e., superset) Q of P. Following [6], we call this operation *subset matching*. The *pattern store* is the data structure used for storing and testing the expandedness of patterns by subset matching. Giunchiglia and Tacchella [6] propose a satisfiability cache based on a bit matrix representation that allows for straightforward subset matching. Practical inputs contain a large number of distinct subformulas, which results in the bit matrix becoming sparse. To exploit this, one can use a sparse matrix representation.

A different data structure for subset and superset matching is proposed by Hoffmann and Koehler [7]. The approach represents patterns as paths in a forest. The forest structure allows sharing of common subpatterns, which can considerably reduce the required space.

3 Lazy Branching

Lazy branching (LB) is a technique that aims at postponing the processing of disjunctions that are consistent with the current tableau branch. LB is inspired by lazy unfolding [8]. Assume a node n contains a disjunction $l \vee t$, where l is a propositional literal. As long as other formulas at n do not constrain l to be false, we can assume l to be true and ignore the disjunction $l \vee t$. In other words, we *delay* the processing of disjunctions for which we know that one of the alternatives (the *witness*) is consistent with the branch. There are two cases in which $l \vee t$ cannot be delayed. Obviously, the disjunction has to be processed if n contains $\bar{l}$, the negation of the witness l. Also, we cannot delay $l \vee t$ if we already delay $\bar{l} \vee s$, since delaying both formulas results in inconsistent assumptions about l. A disjunction $l_1 \vee \cdots \vee l_m \vee t$ with several propositional literals can be delayed as long as at least one of them can serve as a witness.

Propositional literals make good witnesses because their consistency with the branch depends only on formulas at the current node n. A similar observation holds for box formulas. As long as a node n does not contain any diamond formulas $\langle r \rangle t$, it cannot have any r-successors. Hence, all formulas $[r]s$ at n can be assumed true, allowing us to delay disjunctions of the form $[r]s \vee u$.

Compared to lazy unfolding, LB is more general in that it is applicable in more cases. On the other hand, certain special cases are treated more efficiently by lazy unfolding. The specialization of LB to propositional reasoning is known as the Pure Literal Rule. Spartacus implements LB for propositional literals and for box formulas.

4 Evaluation and Conclusion

We evaluate Spartacus on some well known representative benchmarks for **K**: **Table 1:** Randomly generated $3\mathrm{CNF}_K$ formulas [10] for several settings of d (modal depth), L (number of clauses), and N (number of propositional variables). **Table 2:** A subset (the harder problems) of the Tableaux'98 benchmarks for **K** [11]. **Table 3:** A subset (the easier problems) of the TANCS-2000 Unbounded Modal QBF (MQBF) benchmarks [12]. **Table 4:** Randomly generated modalized MQBF formulas [13]. The results for Tables 3 and 4 are grouped by the quantifier alternation depth D and the number of variables V used per alternation in the original QBF. In both cases, the sets contain 8 formulas for each setting of C (number of QBF clauses), which ranges between 10 and 50.

We evaluate the tree (T) and the sparse matrix (S) representation of the pattern store versus no blocking at all (pbb-). Moreover, in one of the runs we disable LB (lb-). The other settings, including the rule application strategy, are chosen according to Configuration I in [9]. To see how Spartacus performs compared to other provers, we include four systems into the evaluation: (1) CWB [14], a prototype reasoner for $\mathcal{ALC}$, featuring global caching. (2) FaCT++ (v1.2.2), currently one of the leading DL reasoners. It supports the logic $\mathcal{SROIQ}$(D), which is more expressive than the language supported by Spartacus. (3) HTab [15] (v1.3.5), a prover for hybrid logic. Compared to Spartacus, HTab additionally supports the difference modality, but has no support for reflexive or transitive relations. (4) *SAT [16] (v1.3), a reasoner for $\mathcal{ALC}$, featuring matrix-based satisfiability and unsatisfiability caching. In contrast to the other systems, which are all tableau-based, *SAT implements a modal extension of the Davis-Putnam procedure. All provers except CWB are compiled and run with the default settings. For CWB we use the flags `-oa -ogc` (global caching on).

The tests are performed on a Pentium 4 2.8GHz, 1GB RAM, with a 60s time limit per formula (60s is enough for most problems). For each setting/system, we count the number of problems solved (left subcolumn). In addition (right subcolumn), we record the average time (in seconds) spent on the successful problems (except for Table 2, where it suffices to give the time for the hardest successful formula). The timings are only relevant for the comparison of two runs if they solve the same number of problems. The best results are set in bold.

d	T		S		pbb-		T,lb-		CWB		FaCT++		HTab		*SAT	
1	45	5.2	45	4.6	45	4.3	45	12.0	0	—	29	32.6	0	—	**45**	**0.4**
2	40	2.5	**40**	**2.4**	**40**	**2.4**	40	3.5	9	1.6	17	5.4	9	1.4	25	9.8
4	38	11.5	**38**	**11.3**	32	12.0	34	12.2	0	—	9	3.2	9	6.2	9	19.9
6	29	19.2	**29**	**17.9**	13	14.9	19	17.0	0	—	0	—	5	15.6	0	—

Table 1: 180 3CNF$_K$ formulas (upper part: 45 formulas with N=5, L=110; lower part: 3×45 formulas with N=3, L=30..150)
Note: LB is effective on hard propositional subproblems. PBB and LB are increasingly effective with growing modal depth.

Test	T		S		pbb-		T,lb-		CWB		FaCT++		HTab		*SAT	
branch_n	9	18.8	9	19.0	8	12.3	8	42.5	**12**	**36.4**	10	48.5	4	4.0	12	50.0
branch_p	11	58.9	11	59.6	8	12.3	8	21.3	**21**	**6.2**	9	11.6	5	58.0	18	57.1
d4_n	**21**	**0.2**	**21**	**0.2**	6	54.4	**21**	**0.2**	21	7.2	21	27.9	6	15.2	**21**	**0.2**
lin_n	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**	21	13.3	21	0.1	21	0.1	13	50.5
path_n	21	0.6	21	0.6	9	50.8	21	0.9	21	51.8	**21**	**0.1**	9	37.1	21	0.2
path_p	21	0.6	21	0.5	10	46.7	21	0.9	21	46.0	**21**	**0.1**	10	36.1	**21**	**0.1**
ph_n	**21**	**1.2**	**21**	**1.2**	**21**	**1.2**	21	4.0	9	33.1	12	20.0	16	30.4	21	1.9
ph_p	8	46.1	8	44.4	8	43.6	8	58.4	7	53.6	7	11.4	6	8.3	**8**	**3.8**

Table 2: Tableaux'98 benchmarks for **K** (8×21 formulas)
Note: PBB is crucial for competitiveness. The rule application strategy chosen for Spartacus is suboptimal for branch_p [9].

V, D	T		S		pbb-		T,lb-		CWB		FaCT++		HTab		*SAT	
4,4	40	0.1	**40**	**0.0**	40	0.1	40	0.1	27	5.7	40	0.4	39	1.1	40	0.1
4,6	**40**	**0.1**	**40**	**0.1**	40	3.0	40	0.2	19	5.8	33	3.2	22	12.5	40	0.8
8,4	40	0.6	**40**	**0.5**	15	9.6	40	1.0	12	7.3	21	6.1	8	11.0	40	8.7
8,6	**39**	**3.7**	38	2.8	9	8.9	36	6.3	8	6.4	15	3.8	4	26.2	26	8.5
16,4	**38**	**2.7**	38	3.6	4	11.8	37	4.6	7	19.0	19	7.6	0	—	24	6.9
16,6	**39**	**1.9**	39	2.3	3	22.4	37	4.4	6	17.5	16	3.4	0	—	24	9.7
4,4	40	4.9	40	5.7	40	5.3	25	30.5	27	30.3	**40**	**1.3**	12	15.5	15	23.9
4,6	4	29.9	3	27.9	4	33.2	1	4.4	1	14.5	**21**	**11.7**	2	16.0	0	—

Table 3: TANCS-2000 benchmarks (upper part: 6×40 `cnfSSS` formulas; lower part: 2×40 `cnfLadn` formulas)
Note: PBB is useful for `cnfSSS`, LB for `cnfLadn`. The rule application strategy chosen for Spartacus is suboptimal for `cnfLadn` [9].

V, D	T		S		pbb-		T,lb-		CWB		FaCT++		HTab		*SAT	
4,4	40	0.4	**40**	**0.3**	4	34.6	40	0.4	13	8.4	39	2.4	0	—	40	0.5
4,6	40	5.2	39	3.6	0	—	38	3.8	8	9.2	25	6.0	0	—	**40**	**5.1**
8,4	33	9.9	**33**	**9.8**	0	—	31	10.3	6	14.1	19	5.0	0	—	26	10.9
8,6	24	3.7	**24**	**3.3**	0	—	24	6.8	0	—	16	7.5	0	—	18	5.4
16,4	22	7.3	**22**	**7.2**	0	—	21	10.2	1	23.4	16	3.8	0	—	17	5.2
16,6	23	6.3	**23**	**5.3**	0	—	21	8.1	0	—	15	3.3	0	—	18	8.2
4,4	**40**	**0.3**	**40**	**0.3**	0	—	**40**	**0.3**	0	—	38	13.3	0	—	40	1.8
4,6	40	0.6	**40**	**0.5**	0	—	40	0.6	0	—	21	26.5	0	—	40	2.9
8,4	**40**	**1.1**	**40**	**1.1**	0	—	40	1.3	0	—	2	40.8	0	—	18	27.0
8,6	**40**	**2.3**	**40**	**2.3**	0	—	40	2.8	0	—	0	—	0	—	18	25.8
16,4	40	5.2	**40**	**4.9**	0	—	40	8.3	0	—	0	—	0	—	0	—
16,6	40	9.9	**40**	**9.8**	0	—	40	18.9	0	—	0	—	0	—	0	—

Table 4: 480 modalized MQBF formulas (upper part: 6×40 `modKSSS` formulas; lower part: 6×40 `modKLadn` formulas)
Note: PBB is crucial. LB is effective on the more complex problems.

The evaluation displays no significant differences in performance between the two implementations of the pattern store, but fully confirms the effectiveness of PBB and LB. The techniques are particularly successful on inputs of high modal depth and on hard propositional subproblems, demonstrating an improvement up to several orders of magnitude. In no case do PBB or LB lead to notable performance penalties. Compared to other systems, the performance of Spartacus proves highly competitive, yielding a promising basis for further research.

References

1. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In Furbach, U., Shankar, N., eds.: IJCAR 2006. Volume 4130 of LNCS., Springer (2006) 292–297
2. Haarslev, V., Möller, R.: RACER system description. In Goré, R., Leitsch, A., Nipkow, T., eds.: IJCAR 2001. Volume 2083 of LNCS., Springer (2001) 701–705
3. Kaminski, M., Smolka, G.: Terminating tableau systems for hybrid logic with difference and converse. To appear in J. Log. Lang. Inf. (2009)
4. Kripke, S.A.: Semantical analysis of modal logic I: Normal modal propositional calculi. Z. Math. Logik Grundlagen Math. **9** (1963) 67–96
5. Kaminski, M., Smolka, G.: Hybrid tableaux for the difference modality. In: 5th Workshop on Methods for Modalities (M4M-5). (2007)
6. Giunchiglia, E., Tacchella, A.: A subset-matching size-bounded cache for testing satisfiability in modal logics. Ann. Math. Artif. Intell. **33**(1) (2001) 39–67
7. Hoffmann, J., Koehler, J.: A new method to index and query sets. In Dean, T., ed.: Proc. 16th Intl. Joint Conf. on Artificial Intelligence (IJCAI'99), Morgan Kaufmann (1999) 462–467
8. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimizing terminological reasoning for expressive description logics. J. Autom. Reasoning **39**(3) (2007) 277–316
9. Götzmann, D.: Spartacus: A Tableau Prover for Hybrid Logic. M.Sc. thesis, Saarland University (2009)
10. Giunchiglia, E., Giunchiglia, F., Tacchella, A.: SAT-based decision procedures for classical modal logics. J. Autom. Reasoning **28**(2) (2002) 143–171
11. Balsiger, P., Heuerding, A.: Comparison of theorem provers for modal logics: Introduction and summary. In de Swart, H., ed.: TABLEAUX'98. Volume 1397 of LNCS. (1998) 25–26
12. Massacci, F., Donini, F.M.: Design and results of TANCS-2000 non-classical (modal) systems comparison. In Dyckhoff, R., ed.: TABLEAUX 2000. Volume 1847 of LNCS., Springer (2000) 52–56
13. Massacci, F.: Design and results of the Tableaux-99 non-classical (modal) systems comparison. In Murray, N.V., ed.: TABLEAUX'99. Volume 1617 of LNCS., Springer (1999) 14–18
14. Goré, R., Postniece, L.: An experimental evaluation of global caching for $\mathcal{ALC}$: System description. In Armando, A., Baumgartner, P., Dowek, G., eds.: IJCAR 2008. Volume 5195 of LNCS., Springer (2008) 299–305
15. Hoffmann, G., Areces, C.: HTab: A terminating tableaux system for hybrid logic. In: 5th Workshop on Methods for Modalities (M4M-5). (2007)
16. Giunchiglia, E., Tacchella, A.: System description: *SAT: A platform for the development of modal decision procedures. In McAllester, D.A., ed.: CADE-17. Volume 1831 of LNCS., Springer (2000) 291–296