Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 262 (2010) 127–139
www.elsevier.com/locate/entcs

Spartacus: A Tableau Prover for Hybrid Logic

Daniel Götzmann[1] Mark Kaminski[1] Gert Smolka[1]

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.

Keywords: hybrid logic, modal logic, tableau algorithms, decision procedures, automated reasoning

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++ [31] and RacerPro [14], 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 [22]. Most of the established blocking techniques are derived from Kripke's chain-based approach [24]. Kaminski and Smolka [21,22] 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 transitive relations. In contrast to other systems, Spartacus uses pattern-based blocking to achieve termination. Similarly to FaCT++ [30,31,32], Spartacus schedules pending rule appli-

[1] Email: {goetzmann,kaminski,smolka}@ps.uni-sb.de

doi:10.1016/j.entcs.2010.04.010

cations 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 simplification (also called "normalization" [17]), Boolean constraint propagation, semantic branching and backjumping [32]. Moreover, Spartacus implements a new technique, called *lazy branching*. Lazy branching is a generalization of lazy unfolding [32], an effective optimization technique from DL reasoning.

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 [12].

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 The Logic

Spartacus decides the satisfiability problem for $\mathcal{H}(E, @)$, the basic hybrid logic extended with global modalities. Notationally, our description of $\mathcal{H}(E, @)$ follows [21]. We distinguish between variables for *states* (x, y), *properties* (p, q), and *relations* (r). From these variables, the *expressions* of $\mathcal{H}(E, @)$ can be obtained by the following grammar:

$$s, t \ ::= \ \dot{\top} \mid p \mid \dot{x} \mid \dot{\neg} s \mid s \mathbin{\dot{\wedge}} s \mid \langle r \rangle s \mid Es \mid @xs$$

We employ the usual abbreviations $s \mathbin{\dot{\vee}} t := \dot{\neg}(\dot{\neg} s \mathbin{\dot{\wedge}} \dot{\neg} t)$, $[r]s := \dot{\neg}\langle r \rangle \dot{\neg} s$, and $As := \dot{\neg} E \dot{\neg} s$. For details on $\mathcal{H}(E, @)$ and related logics, see [1].

In addition to expressions of the above form, Spartacus accepts reflexivity and transitivity assertions of the form *Reflexive* r and *Transitive* r.

Except for the details of the blocking mechanism, the calculus underlying Spartacus is a restriction of the system in [22] to $\mathcal{H}(E, @)$. The calculus works on formulas of the form sx where s is a negation normal expression of $\mathcal{H}(E, @)$ and x a state. The use of the state variable x in a formula sx corresponds to the use of *prefixes* [6] or *nodes* [19] in related calculi. Since for later discussion the treatment of equality in Spartacus is inessential, let us consider the following restriction of the calculus to **K**.

$$\mathcal{R}_{\dot{\neg}} \ \frac{px,\ (\dot{\neg} p)x}{\bot} \qquad \mathcal{R}_{\dot{\wedge}} \ \frac{(s \mathbin{\dot{\wedge}} t)x}{sx,\ tx} \qquad \mathcal{R}_{\dot{\vee}} \ \frac{(s \mathbin{\dot{\vee}} t)x}{sx \ \mid \ tx} \qquad \mathcal{R}_{\Diamond} \ \frac{(\langle r \rangle s)x}{rxy,\ sy}\ y \text{ fresh} \qquad \mathcal{R}_{\Box} \ \frac{([r]s)x,\ rxy}{sy}$$

The symbol $\bot$ marks closed branches. The formula rxy specifies that y has to be *accessible* from x, corresponding to the notation $x \Diamond_r y$ in [6] and $\langle x, y \rangle \in \mathcal{E}_A(r)$ in [19].

3 Pattern-Based Blocking

Pattern-based blocking (PBB) in Spartacus is implemented following [21]. The technique yields termination in the presence of nominals, transitive relations, global

and difference modalities. To deal with graded modalities, PBB can be extended as proposed in [20,23]. These extensions, however, are currently not supported by Spartacus.

Like traditional blocking techniques, PBB is an applicability restriction on the diamond rule $\mathcal{R}_\Diamond$. Let $R_x r := \{[r]s \mid ([r]s)x \text{ is on the branch}\}$. PBB as formulated in [21] restricts $\mathcal{R}_\Diamond$ to only be applicable to a formula $(\langle r\rangle s)x$ if there are no states x', y such that $rx'y$ and sy are on the branch, and $R_x r \subseteq R_{x'} r$.

The *pattern* of a diamond expression $\langle r\rangle s$ at a state x is defined as $P_x(\langle r\rangle s) := \{\langle r\rangle s\} \cup R_x r$. Let us reformulate the blocking condition to make explicit how diamond patterns come into play here. Let $(\langle r\rangle s)x$ be on the branch. We say the pattern $P_x(\langle r\rangle s)$ is *expanded* if there are states x', y such that $rx'y$ and sy are on the branch, and $R_x r \subseteq R_{x'} r$. Then we can say $\mathcal{R}_\Diamond$ is applicable to a formula $(\langle r\rangle s)x$ if the pattern $P_x(\langle r\rangle s)$ is not expanded.

Clearly, after $\mathcal{R}_\Diamond$ has been applied to a formula $(\langle r\rangle s)x$, the pattern $P_x(\langle r\rangle s)$ becomes expanded. Note that while the notation $P_x(\langle r\rangle s)$ depends on the state x of the current tableau branch, generally patterns are just sets of expressions. Given two patterns P, Q, it is easy to check that if $P \subseteq Q$ and Q is expanded, then so is P. Additionally, calculi like [6,22] enjoy the property that once a pattern is expanded on a tableau branch, it will stay expanded on all extensions of the branch. These considerations suggest an efficient algorithmic implementation of (a slightly weakened version of) the blocking condition.

Every time we want to apply $\mathcal{R}_\Diamond$ to a formula sx, we need to check if $P_x s$ is expanded. Instead of computing this information from the branch we use a special data structure, called the *pattern store*, to record and query which patterns are expanded. The pattern store contains all patterns that are known to be expanded because of previous diamond rule applications. Whenever $\mathcal{R}_\Diamond$ is applied to a formula sx, the pattern $P_x s$ is added to the pattern store. When checking if $\mathcal{R}_\Diamond$ applies to a formula sx, we have to check if the pattern store contains a superset of $P_x s$. The efficiency of this operation, called *subset matching* following [9], is crucial for the performance of PBB.

Giunchiglia and Tacchella [9] propose a satisfiability cache based on a bit matrix representation that allows for straightforward subset matching. Practical inputs contain a large number of distinct subexpressions, which results in the bit matrix becoming sparse. To exploit this, one uses a sparse matrix representation. A different data structure for subset and superset matching is proposed by Hoffmann and Koehler [16]. The approach represents patterns as paths in a forest. The forest structure allows sharing of common subpatterns, which can considerably reduce the required space. We implemented both data structures for subset matching so as to be able to compare their performance. From an implementation point of view, the forest-based structure by Hoffman and Koehler turned out to be more challenging than the matrix-based structure. Their evaluation in the present setting, however, revealed no significant differences in performance. The default configuration of Spartacus uses the structure by Hoffmann and Koehler.

As an optimization of the basic implementation of blocking described above, we

also add the pattern $P_x(\langle r\rangle s)$ to the pattern store whenever the branch is extended by a formula of the form $([r]t)x$, provided $\mathcal{R}_\Diamond$ has already been applied to $(\langle r\rangle s)x$ before and the store contains no superset of $P_x(\langle r\rangle s)$.

What are the differences of PBB from other blocking techniques? Conceptually, the main difference is that the blocking relation in PBB is defined on patterns rather than states. Given a state x, the *label set* of x, $\{s \mid sx \text{ is on the branch}\}$, may contain arbitrary expressions. A pattern, on the other hand, always consists of exactly one diamond and a set of boxes. With other blocking techniques, the number of states generated on a branch is bounded by the number of possible label sets, i.e., subsets of the subterm closure of the input expression. In PBB, the corresponding bound is the number of patterns contained in the subterm closure of the input. It is easily seen that the latter number is often much smaller than the former number: A subterm closure containing m (distinct) diamonds and n boxes has at least cardinality $m+n+1$ (typically larger). Hence, the closure contains $m\cdot 2^n$ patterns but over 2^{m+n} label sets. This suggests that PBB is likely to terminate faster and produce smaller models compared to techniques working with label sets. Other than that, PBB most closely resembles *anywhere blocking* as described by Baader et al. [2], and shares the advantages of anywhere blocking over the more traditional ancestor-based techniques of [24,19,6,22]. In particular, PBB can reduce the size of a tableau derivation even in the context of **K**, where ancestor-based techniques have no effect.

The implementation of PBB in Spartacus is inspired by modal caching techniques, in particular the one described by Giunchiglia and Tacchella [9]. In fact, the sets of expressions that are considered in [9] are nothing other than patterns, and the satisfiability cache of [9] provides exactly the kind of storage and lookup operations that are also necessary for PBB. So how does PBB differ from satisfiability caching in [9]? While PBB subsumes satisfiability caching, the converse is not true. In particular, the system of [9] does not terminate in the presence of transitivity or global modalities and hence needs to be complemented by a blocking technique. To retain completeness in the presence of blocking, however, satisfiability caching needs to be refined considerably [13].

Yet another approach to termination is global caching [10,11]. Global caching combines properties of anywhere blocking, satisfiability and unsatisfiability caching. Rather than looking at a single tableau branch at a time, global caching incrementally constructs the entire tableau (as an and-or graph), which allows to naturally re-use intermediate satisfiability and unsatisfiability results. Global caching has potential advantages as well as potential disadvantages compared to PBB. An advantage is the ability to re-use unsatisfiability results from previous branches, which is not provided by PBB, but can potentially be obtained by combining PBB with an unsatisfiability cache. A potential disadvantage of global caching and unsatisfiability caching alike is the higher memory consumption, which may have an adverse effect on performance [18].

4 Lazy Branching

Lazy branching (LB) is a technique that dynamically reorders the processing of disjunctions, aiming at a more goal directed exploration of the search space. Conceptually, LB is a rule application strategy, somewhat unusual in that it may avoid a rule application forever. The idea is to avoid the processing of disjunctions that are consistent with the current tableau branch. LB is inspired by lazy unfolding [3,32].

Lazy unfolding aims at improving DL reasoning with respect to TBoxes. Using our present formalism, a TBox $\mathcal{T}$ can be seen as a set of expressions of the form As. To test if an expression s is *consistent with respect to* $\mathcal{T}$, one tests if the conjunction $s \mathbin{\dot\wedge} \dot\bigwedge_{t\in\mathcal{T}} t$ is satisfiable. The naive idea to treat a TBox $\mathcal{T}$ is to add tx to the tableau branch for every state x on the branch and every expression $At \in \mathcal{T}$. Lazy unfolding provides a better treatment of TBoxes (or parts of TBoxes), provided the (sub-)TBox is unfoldable. A TBox $\mathcal{T}$ is *unfoldable* if:

(i) Every expression in $\mathcal{T}$ is of the form $A(p \mathrel{\dot\to} s)$ or $A(p \mathrel{\dot\leftrightarrow} s)$.

(ii) If $A(p \mathrel{\dot\leftrightarrow} s) \in \mathcal{T}$, then $\mathcal{T}$ contains no expressions $A(p \mathrel{\dot\leftrightarrow} t)$ or $A(p \mathrel{\dot\to} t)$.

(iii) Expressions $A(p \mathrel{\dot\leftrightarrow} s) \in \mathcal{T}$ satisfy a certain acyclicity condition, essentially meaning that p does not occur in s (see [32] for details).

Let us restrict our attention to expressions of the form $A(p \mathrel{\dot\to} s)$; expressions $A(p \mathrel{\dot\leftrightarrow} s)$ are treated in essentially the same way. If $A(p \mathrel{\dot\to} s)$ is part of an unfoldable TBox $\mathcal{T}$ and px is on the branch, then lazy unfolding extends the branch by sx. This can be seen as a unit resolution step between px and the disjunction $(\dot\neg p \mathbin{\dot\vee} s)x$ that would have been added to the branch by the naive treatment, only that the actual addition never takes place.

LB generalizes lazy unfolding in that it applies to propositional literals (i.e., possibly negated properties) within arbitrary disjunctions, not just disjunctions coming from unfoldable expressions of the form $A(p \mathrel{\dot\to} s)$ and $A(p \mathrel{\dot\leftrightarrow} s)$. Also, the approach easily generalizes from propositional literals to other "simple" expressions such as boxes. The idea for propositional literals is as follows. Assume the branch contains a disjunction $(l \mathbin{\dot\vee} s)x$ where l is a propositional literal. As long as there are no formulas on the branch that constrain lx to be false, we can assume lx to be true and ignore the disjunction $(l \mathbin{\dot\vee} s)x$. 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 $(l \mathbin{\dot\vee} s)x$ cannot be delayed. Obviously, the disjunction has to be processed if the branch contains $\bar{l}x$, the negation of the witness lx. Also, we cannot delay $(l \mathbin{\dot\vee} s)x$ if we already delay $(\bar{l} \mathbin{\dot\vee} t)x$, since delaying both formulas results in inconsistent assumptions about the truth value of lx. A disjunction $(l_1 \mathbin{\dot\vee} \ldots \mathbin{\dot\vee} l_m \mathbin{\dot\vee} s)x$ 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 can be checked "locally" within a label set. A similar observation holds for box expressions. As long as the branch contains no formulas $(\langle r\rangle s)x$, x does not need to have any r-successors. Hence, all formulas $([r]t)x$ can be assumed true,

allowing us to delay disjunctions of the form $([r]t \dot{\vee} s')x$.

Compared to lazy unfolding, LB is more general in that it is applicable in more cases. On the other hand, in cases where both techniques apply, lazy unfolding is likely to be more effective. This is because, rather than restricting the processing of delayed disjunctions, it does not generate such disjunctions in the first place.

LB for propositional literals and boxes is implemented as an additional layer on top of the rule application queue. While conventional rule application heuristics (as in [30]) influence the position of a pending rule application in the queue, LB prevents disjunctions from being added to the queue as long as they are delayed. This allows LB to work independently of the conventional rule application strategy.

5 Evaluation

We evaluate the effects of PBB and LB by comparing the performance of the default configuration of Spartacus with two modified configurations where we switch off PBB and LB, respectively. The default configuration applies $\mathcal{R}_\vee$ with the lowest priority, in particular with a lower priority than $\mathcal{R}_\Diamond$. This differs from the default rule application strategy in many other provers, like FaCT++ or *SAT [8], which apply the diamond rule with the lowest priority. Therefore, we also include in the tests a configuration where $\mathcal{R}_\Diamond$ is applied with the lowest priority ("$\Diamond$ last"), which is achieved by the option `--exp-ord="[@nAE|<"`. To see how Spartacus (v1.0.1) performs compared to other provers, we include four systems into the evaluation:

- A prototype prover for propositional dynamic logic (PDL) by Goré and Widmann [11]. The prover implements a worst-case optimal decision procedure for PDL featuring global caching. In the tables, we refer to it as "pdl".
- FaCT++ (v1.3.0), currently one of the leading DL reasoners. It supports the logic $\mathcal{SROIQ}$(D), which is more expressive than the language supported by Spartacus. FaCT++ implements anywhere blocking.
- HTab [15] (v1.4.0), a prover for hybrid logic. Compared to Spartacus, HTab additionally supports the difference modality, but has no support for reflexive or transitive relations. HTab implements ancestor-based blocking.
- *SAT [8] (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. *SAT uses no blocking mechanism.

All provers are compiled and run with the default settings. Unfortunately, we were not able to include into the comparison the prover DLP [27], reportedly one of the fastest provers for **K**. The reason is that DLP relies on an outdated version of the SML/NJ compiler that we were not able to install on our test machine. To get an impression of the performance of DLP, refer to [28,7].

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 problem). 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. In the figures, we plot the number of instances that could be solved from a given class of problems against time. Each figure consists of two plots, the one on the left-hand side comparing the performance of different configurations of Spartacus, and the one on the right-hand side comparing the default configuration of Spartacus against other provers.

We restrict our attention to **K** since PBB and LB are not specific to hybrid logic, but the selection of available benchmarks and provers is much larger for **K** than for $\mathcal{H}(@)$ or $\mathcal{H}(E, @)$. For an evaluation of Spartacus on $\mathcal{H}(E, @)$, see [12]; a more detailed evaluation is reserved for future work.

We use the following well-known benchmarks:

Table 1, Fig. 1, 2: Randomly generated 3CNF_K formulas [7] for several settings of d (modal depth), L (number of clauses), and N (number of propositional variables). We group the problems in two subclasses according to d, namely one subclass with $d = 1$ and one with $d = 2$, 4, and 6. The former class is generated with $N = 5$, $L = 110$, and $p = 0$, where p is the probability of a disjunct occurring at depth $< d$ being purely propositional (see [7] for details). The ratio of satisfiable to unsatisfiable instances is 14/31, which allows us to test the behaviour in the presence of backtracking. The latter class is designed to test the behaviour on formulas of increasingly high modal depth. For each setting of d, the problems consist of 9 formulas for $L = 30$, 60, 90, 120, and 150, respectively. In all cases, N is set to 3 and p to 0. To keep the complexity manageable, we restrict ourselves to satisfiable problems.

Table 2: A subset of the Tableaux'98 benchmark suite for **K** [4,5]. The suite consists of 9 satisfiable and 9 unsatisfiable classes, each consisting of 21 problems of increasing size and difficulty. We do not display the results for the classes `dum_n`, `dum_p`, and `grz_n` because these classes can be solved in the hardest instance by all configurations of Spartacus and all the other provers that we consider.

Table 3, Fig. 3, 4: A subset (the easier problems) of the TANCS-2000 Unbounded Modal QBF (MQBF) benchmarks for **K** [26].

Table 4, Fig. 5, 6: Randomly generated modalized MQBF formulas [25]. Modalization aims at reducing the influence of propositional optimizations by encoding different propositional variables as different modal subexpressions.

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.

Compared to the other problem sets, 3CNF_K with $d = 1$ (Table 1, Fig. 1) contains many hard propositional subproblems and requires a lot of backtracking. With LB we observe a noticeable speedup. PBB, on the other hand, shows no positive effect, which we believe is due to the low modal depth of the problems.

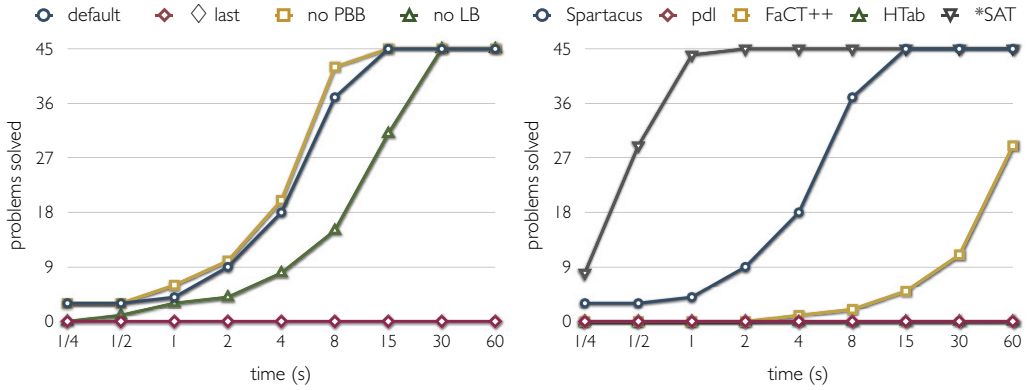

Fig. 1. $3\mathrm{CNF}_K$ $d = 1$

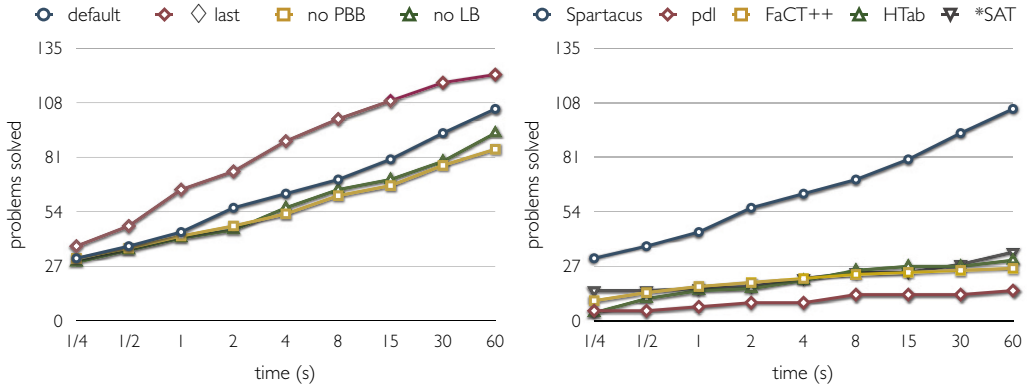

Fig. 2. $3\mathrm{CNF}_K$ $d = 2..6$

Compared to the other provers, Spartacus is second only to *SAT. *SAT, being based on a SAT solver engine, is expectedly successful on problems dominated by propositional reasoning. On problems with $d = 2$, 4, and 6 (Table 1, Fig. 2), we can clearly see how the effect of PBB increases with growing modal depth. Similarly if not more influential is, however, the chosen rule application strategy.

d	default		◇ last		no PBB		no LB		pdl		FaCT++		HTab		*SAT	
1	45	5.2	0	—	45	4.3	45	12.0	0	—	29	33.1	0	—	**45**	**0.4**
2	40	2.5	32	2.7	**40**	**2.4**	40	3.5	7	1.3	17	5.4	12	5.6	25	9.8
4	38	11.5	**45**	**0.9**	32	12.0	34	12.2	7	9.3	9	3.1	10	4.8	9	19.9
6	27	16.1	**45**	**10.6**	13	14.9	19	17.0	1	37.9	0	—	8	6.3	0	—

Table 1
$3\mathrm{CNF}_K$ (upper part: 45 formulas; lower part: 3×45 formulas)

On the Tableaux'98 benchmarks (Table 2), one can see that PBB is crucial for the competitiveness of Spartacus, having a decisive influence on the results for `d4_n`, `path_n`, and `path_p`. In fact, the only prover that does not perform well on these three problems is HTab, i.e., the only prover that has no effective blocking or caching technique for **K**. On `branch_p`, the default strategy shows inferior to applying $\mathcal{R}_\Diamond$ last. Note that in [7], *SAT is reported to display a better performance

on `branch_p`, `lin_n` and `ph_n` than we observe here. We believe this to be the case because the evaluation in [7] uses a version of *SAT compiled with non-default settings specifically tailored for the Tableaux'98 benchmark suite (see [29]).

Test	default		◇ last		no PBB		no LB		pdl		FaCT++		HTab		*SAT	
branch_n	9	18.8	10	14.8	8	12.3	8	42.5	7	22.9	10	49.2	8	20.8	**12**	**50.0**
branch_p	11	58.9	16	26.8	8	12.3	8	21.3	4	8.8	9	11.6	8	18.5	**18**	**57.1**
d4_n	21	0.2	21	0.6	6	54.4	21	0.2	**21**	**0.1**	21	27.7	6	57.5	21	0.2
d4_p	21	0.1	21	0.1	21	0.3	21	0.1	21	0.1	21	0.1	18	59.3	**21**	**0.0**
grz_p	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**	14	27.0	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**
lin_n	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**	21	0.1	21	0.1	21	0.1	13	50.5
lin_p	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**	7	9.7	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**
path_n	21	0.6	21	0.6	9	50.8	21	0.9	21	0.4	**21**	**0.1**	8	27.6	21	0.2
path_p	21	0.6	21	0.5	10	46.7	21	0.9	21	0.4	**21**	**0.1**	9	35.9	**21**	**0.1**
ph_n	**21**	**1.2**	**21**	**1.2**	**21**	**1.2**	21	4.0	7	5.1	12	18.8	16	39.5	11	23.8
ph_p	8	46.1	8	42.6	8	43.6	8	58.4	5	6.3	7	12.3	6	10.1	**8**	**3.8**
poly_n	21	0.2	21	0.2	21	4.7	21	0.2	9	45.6	**21**	**0.1**	21	6.4	**21**	**0.1**
poly_p	21	0.2	**21**	**0.1**	21	4.5	21	0.3	8	17.7	**21**	**0.1**	21	11.4	**21**	**0.1**
t4p_n	21	0.1	**21**	**0.0**	21	0.1	21	0.5	21	0.1	21	0.3	4	16.8	21	0.1
t4p_p	**21**	**0.0**	**21**	**0.0**	**21**	**0.0**	21	0.2	21	0.1	21	0.1	6	32.4	**21**	**0.0**

Table 2
Tableaux'98 benchmarks for **K** (15×21 formulas)

On the TANCS-2000 benchmarks (Table 3), PBB proves very successful on `cnfSSS` (Fig. 3) while LB yields a notable speedup on `cnfLadn` (Fig. 4). The increase in effectiveness of LB on `cnfLadn` compared to `cnfSSS` correlates well with the fact that `cnfLadn` requires significantly more backtracking than `cnfSSS`. Also, applying $\mathcal{R}_\Diamond$ last significantly improves the performance of Spartacus on `cnfLadn`. While being behind pdl and FaCT++ with respect to the number of problems solved with the default strategy, Spartacus achieves the best result with the alternative strategy.

V, D	default		◇ last		no PBB		no LB		pdl		FaCT++		HTab		*SAT	
4,4	40	0.1	**40**	**0.0**	40	0.1	40	0.1	28	5.2	30	0.1	39	2.3	40	0.1
4,6	**40**	**0.1**	**40**	**0.1**	40	3.0	40	0.2	27	3.3	21	0.9	17	10.2	40	0.8
8,4	40	0.6	**40**	**0.2**	15	9.6	40	1.0	25	3.8	12	0.1	7	11.5	40	8.7
8,6	39	3.7	**40**	**1.3**	9	8.9	36	6.3	25	3.7	13	1.5	3	21.4	26	8.5
16,4	38	2.7	**40**	**2.0**	4	11.8	37	4.6	29	7.7	14	1.8	0	—	24	6.9
16,6	39	1.9	**40**	**1.3**	3	22.4	37	4.4	27	9.7	15	1.8	0	—	24	9.7
4,4	40	4.9	40	2.4	40	5.3	25	30.5	28	13.7	**40**	**1.3**	7	16.9	15	23.9
4,6	4	29.9	**23**	**14.1**	4	33.2	1	4.4	17	13.1	21	11.8	1	48.1	0	—

Table 3
TANCS-2000 benchmarks for **K** (upper part: 6×40 `cnfSSS` formulas; lower part: 2×40 `cnfLadn` formulas)

Modalized MQBF problems (Table 4, Fig. 5, 6) have a higher modal depth than their non-modalized counterparts, which suggests a potentially higher effectiveness of PBB. And indeed, PBB proves highly successful on modalized formulas. While Spartacus can solve almost none of the problems without PBB, with PBB it handles most of them easily. Since the problems can be solved without much backtracking, the effectiveness of LB is limited. Note also that applying $\mathcal{R}_\Diamond$ last, while being superior to the default strategy on `modKSSS`, is largely ineffective on `modKLadn`.

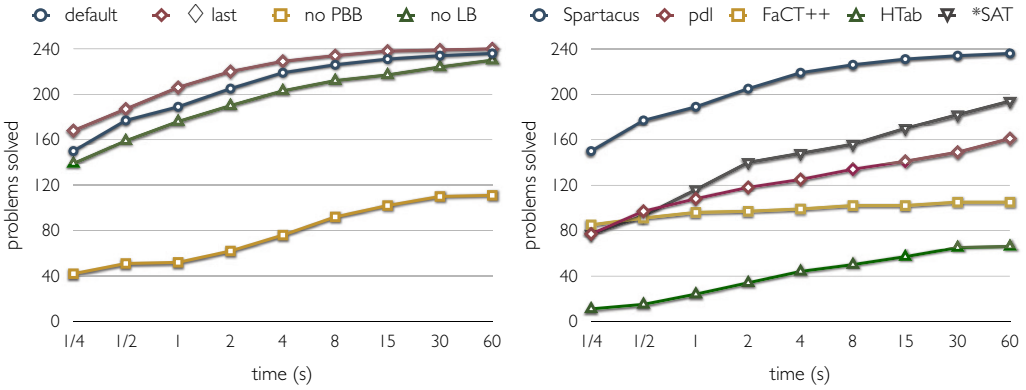

Fig. 3. TANCS-2000: `cnfSSS`

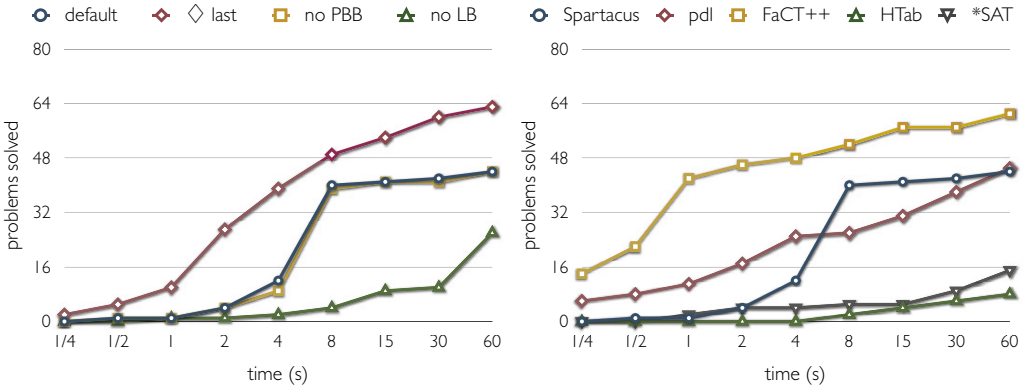

Fig. 4. TANCS-2000: `cnfLadn`

V, D	default		◇ last		no PBB		no LB		pdl		FaCT++		HTab		*SAT	
4,4	40	0.4	**40**	**0.2**	4	34.6	40	0.4	19	3.0	21	0.1	0	—	40	0.5
4,6	40	5.2	**40**	**0.8**	0	—	38	3.8	17	1.3	9	0.1	0	—	40	5.1
8,4	33	9.9	**40**	**5.1**	0	—	31	10.3	21	2.4	11	0.5	0	—	26	10.9
8,6	24	3.7	**37**	**8.8**	0	—	24	6.8	21	4.5	13	3.8	0	—	18	5.4
16,4	22	7.3	**33**	**7.4**	0	—	21	10.2	21	4.2	15	4.0	0	—	17	5.2
16,6	23	6.3	**30**	**5.3**	0	—	21	8.1	21	8,4	13	3.1	0	—	18	8.2
4,4	**40**	**0.3**	5	32.1	0	—	**40**	**0.3**	0	—	38	13.2	0	—	40	1.8
4,6	**40**	**0.6**	0	—	0	—	**40**	**0.6**	0	—	21	26.5	0	—	40	2.9
8,4	**40**	**1.1**	0	—	0	—	40	1.3	0	—	2	40.2	0	—	18	27.0
8,6	**40**	**2.3**	0	—	0	—	40	2.8	0	—	0	—	0	—	18	25.8
16,4	**40**	**5.2**	0	—	0	—	40	8.3	0	—	0	—	0	—	0	—
16,6	**40**	**9.9**	0	—	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)

6 Conclusion

The evaluation confirms the effectiveness of PBB on a wide range of problems. On 3CNF_K the effect of PBB can be seen growing with increasing modal depth. On some problems from the Tableaux'98 suite, on `cnfSSS` from TANCS-2000, and on

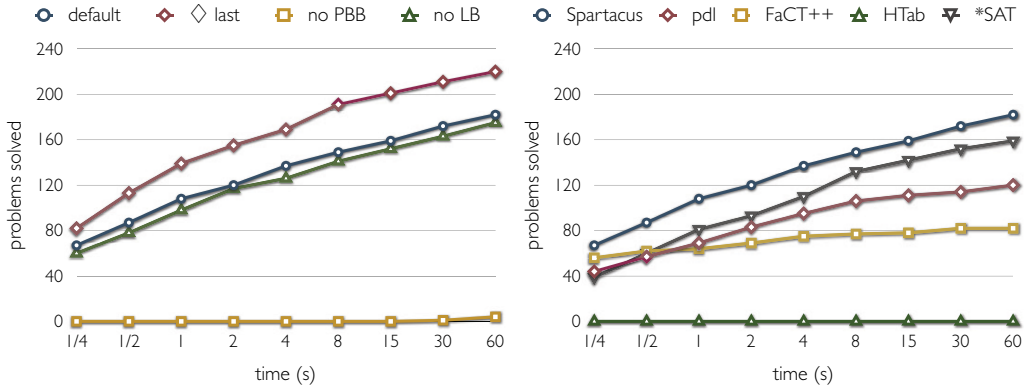

Fig. 5. Modalized MQBF: `modKSSS`

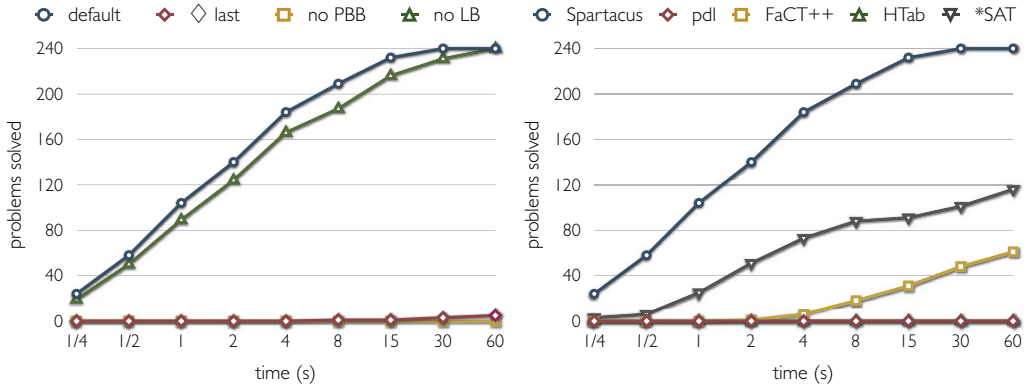

Fig. 6. Modalized MQBF: `modKLadn`

the modalized problems, PBB demonstrates an improvement up to several orders of magnitude. The influence of LB is less noticeable on the majority of the problems. Given that LB is a technique for optimizing the exploration of the search space, this is not surprising since most of the problems we consider require no or little backtracking. On the two problem classes that do require a lot of backtracking, 3CNF_K with $d = 1$ and `cnfLadn` from TANCS-2000, LB proves quite successful. In no case do PBB or LB lead to notable performance penalties.

A major factor influencing the performance of Spartacus is the chosen rule application strategy. Giving $\mathcal{R}_\Diamond$ the lowest priority strongly boosts performance on some of the benchmarks, even surpassing the effect of PBB on 3CNF_K with $d \geq 4$, on `branch_p` from Tableaux'98, and on `cnfLadn` from TANCS-2000. At the same time, the strategy leads to an even more significant degradation of performance on 3CNF_K with $d = 1$ and `modKLadn`. Since the default rule application strategy displays a reasonable performance on all of the benchmarks, overall, it appears more attractive. Still, as noted by Tsarkov and Horrocks [30], finding an adequate rule application strategy is of crucial importance for the performance of modal tableau algorithms. Devising adaptive strategies that would be competitive on a wide range of inputs remains a challenging open problem.

Compared to other systems, the performance of Spartacus proves highly compet-

itive, yielding a promising basis for further research. One interesting direction would be extending Spartacus to more expressive logics. Some constructs, like the difference modality, can be integrated within the existing architecture. Graded modalities and role hierarchies could be integrated by extending PBB following [20,23]. Since PBB does not support converse modalities, extending the system to handle converse would require complementing PBB by other techniques.

References

[1] Areces, C. and B. ten Cate, *Hybrid logics*, in: P. Blackburn, J. van Benthem and F. Wolter, editors, *Handbook of Modal Logic*, Studies in Logic and Practical Reasoning **3**, Elsevier, 2006 pp. 821–868.

[2] Baader, F., M. Buchheit and B. Hollunder, *Cardinality restrictions on concepts*, Artif. Intell. **88** (1996), pp. 195–213.

[3] Baader, F., B. Hollunder, B. Nebel, H.-J. Profitlich and E. Franconi, *An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on*, Appl. Intell. **4** (1994), pp. 109–132.

[4] Balsiger, P. and A. Heuerding, *Comparison of theorem provers for modal logics: Introduction and summary*, in: H. de Swart, editor, *TABLEAUX'98*, LNCS **1397**, 1998, pp. 25–26.

[5] Balsiger, P., A. Heuerding and S. Schwendimann, *A benchmark method for the propositional modal logics K, KT, S4*, J. Autom. Reasoning **24** (2000), pp. 297–317.

[6] Bolander, T. and P. Blackburn, *Termination for hybrid tableaus*, J. Log. Comput. **17** (2007), pp. 517–554.

[7] Giunchiglia, E., F. Giunchiglia and A. Tacchella, *SAT-based decision procedures for classical modal logics*, J. Autom. Reasoning **28** (2002), pp. 143–171.

[8] Giunchiglia, E. and A. Tacchella, *System description: *SAT: A platform for the development of modal decision procedures*, in: D. A. McAllester, editor, *CADE-17*, LNCS **1831** (2000), pp. 291–296.

[9] Giunchiglia, E. and A. Tacchella, *A subset-matching size-bounded cache for testing satisfiability in modal logics*, Ann. Math. Artif. Intell. **33** (2001), pp. 39–67.

[10] Goré, R. and L. A. Nguyen, *EXPTIME tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies*, in: N. Olivetti, editor, *TABLEAUX'07*, LNCS **4548** (2007), pp. 133–148.

[11] Goré, R. and F. Widmann, *An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability*, in: R. A. Schmidt, editor, *CADE-22*, LNCS (LNAI) **5663** (2009), pp. 437–452.

[12] Götzmann, D., "Spartacus: A Tableau Prover for Hybrid Logic," M.Sc. thesis, Saarland University (2009).

[13] Haarslev, V. and R. Möller, *Consistency testing: The RACE experience*, in: R. Dyckhoff, editor, *TABLEAUX 2000*, LNCS **1847** (2000), pp. 57–61.

[14] Haarslev, V. and R. Möller, *RACER system description*, in: R. Goré, A. Leitsch and T. Nipkow, editors, *IJCAR 2001*, LNCS **2083** (2001), pp. 701–705.

[15] Hoffmann, G. and C. Areces, *HTab: A terminating tableaux system for hybrid logic*, in: C. Areces and S. Demri, editors, *Proc. 5th Workshop on Methods for Modalities (M4M5 2007)*, Electr. Notes Theor. Comput. Sci. **231** (2009), pp. 3–19.

[16] Hoffmann, J. and J. Koehler, *A new method to index and query sets*, in: T. Dean, editor, *Proc. 16th Intl. Joint Conf. on Artificial Intelligence (IJCAI'99)* (1999), pp. 462–467.

[17] Horrocks, I., *Implementation and optimization techniques*, in: F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press, 2007, 2nd edition pp. 329–373.

[18] Horrocks, I. and P. F. Patel-Schneider, *Optimizing description logic subsumption*, J. Log. Comput. **9** (1999), pp. 267–293.

[19] Horrocks, I. and U. Sattler, *Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic*, in: B. Nebel, editor, *Proc. 17th Intl. Joint Conf. on Artificial Intelligence (IJCAI 2001)* (2001), pp. 199–204.

[20] Kaminski, M., S. Schneider and G. Smolka, *Terminating tableaux for graded hybrid logic with global modalities and role hierarchies*, in: M. Giese and A. Waaler, editors, *TABLEAUX 2009*, LNCS (LNAI) **5607** (2009), pp. 235–249.

[21] Kaminski, M. and G. Smolka, *Hybrid tableaux for the difference modality*, in: C. Areces and S. Demri, editors, *Proc. 5th Workshop on Methods for Modalities (M4M5 2007)*, Electr. Notes Theor. Comput. Sci. **231** (2009), pp. 241–257.

[22] Kaminski, M. and G. Smolka, *Terminating tableau systems for hybrid logic with difference and converse*, J. Log. Lang. Inf. **18** (2009), pp. 437–464.

[23] Kaminski, M. and G. Smolka, *Terminating tableaux for $\mathcal{SOQ}$ with number restrictions on transitive roles*, in: B. C. Grau, I. Horrocks, B. Motik and U. Sattler, editors, *Proc. 22nd Intl. Workshop on Description Logics (DL 2009)*, CEUR Workshop Proceedings **477**, 2009.

[24] Kripke, S. A., *Semantical analysis of modal logic I: Normal modal propositional calculi*, Z. Math. Logik Grundlagen Math. **9** (1963), pp. 67–96.

[25] Massacci, F., *Design and results of the Tableaux-99 non-classical (modal) systems comparison*, in: N. V. Murray, editor, *TABLEAUX'99*, LNCS **1617** (1999), pp. 14–18.

[26] Massacci, F. and F. M. Donini, *Design and results of TANCS-2000 non-classical (modal) systems comparison*, in: R. Dyckhoff, editor, *TABLEAUX 2000*, LNCS **1847** (2000), pp. 52–56.

[27] Patel-Schneider, P. F., *System description: DLP*, in: D. McAllester, editor, *CADE-17*, LNCS **1831** (2000), pp. 297–301.

[28] Patel-Schneider, P. F., *TANCS-2000 results for DLP*, in: R. Dyckhoff, editor, *TABLEAUX 2000*, LNCS **1847** (2000), pp. 72–76.

[29] Tacchella, A., **SAT user's manual, version 1.3* (2000), draft.
URL www.mrg.dist.unige.it/~tac/StarSAT/manual.ps.gz

[30] Tsarkov, D. and I. Horrocks, *Ordering heuristics for description logic reasoning*, in: L. P. Kaelbling and A. Saffiotti, editors, *Proc. 19th Intl. Joint Conf. on Artificial Intelligence (IJCAI'05)* (2005), pp. 609–614.

[31] Tsarkov, D. and I. Horrocks, *FaCT++ description logic reasoner: System description*, in: U. Furbach and N. Shankar, editors, *IJCAR 2006*, LNCS **4130** (2006), pp. 292–297.

[32] Tsarkov, D., I. Horrocks and P. F. Patel-Schneider, *Optimizing terminological reasoning for expressive description logics*, J. Autom. Reasoning **39** (2007), pp. 277–316.