Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

A SAT-Based Approach for Solving the Modal Logic S5-Satisfiability Problem

Thomas Caridroit, Jean-Marie Lagniez, Daniel Le Berre, Tiago de Lima and **Valentin Montmirail**

CRIL, Univ. Artois and CNRS, F62300 Lens, France
{caridroit,lagniez,leberre,delima,montmirail}@cril.fr

Abstract

We present a SAT-based approach for solving the modal logic S5-satisfiability problem. That problem being NP-complete, the translation into SAT is not a surprise. Our contribution is to greatly reduce the number of propositional variables and clauses required to encode the problem. We first present a syntactic property called diamond degree. We show that the size of an S5-model satisfying a formula ϕ can be bounded by its diamond degree. Such measure can thus be used as an upper bound for generating a SAT encoding for the S5-satisfiability of that formula. We also propose a lightweight caching system which allows us to further reduce the size of the propositional formula. We implemented a generic SAT-based approach within the modal logic S5 solver S52SAT. It allowed us to compare experimentally our new upper-bound against previously known one, i.e. the number of modalities of ϕ and to evaluate the effect of our caching technique. We also compared our solver against existing modal logic S5 solvers. The proposed approach outperforms previous ones on the benchmarks used. These promising results open interesting research directions for the practical resolution of others modal logics (e.g. K, KT, S4)

Introduction

Over the last twenty years, modal logics have been used in various areas of artificial intelligence like formal verification (Fairtlough and Mendler 1994), game theory (Lorini and Schwarzentruber 2010), database theory (Fitting 2000) and distributed computing (VII, Crary, and Harper 2005) for example. More recently, the modal logic S5 was used for contingent planning (Niveau and Zanuttini 2016) and in knowledge compilation (Bienvenu, Fargier, and Marquis 2010). For this reason, automated reasoning in modal logics has been vastly studied (e.g. (Sebastiani and Villafiorita 1998; Masssacci 2000; Sebastiani and Vescovi 2009)).

Ladner (Ladner 1977; Fagin et al. 1995) showed that the satisfiability problem for several modal logics including K, KT and S4 is PSPACE-Complete while it is NP-Complete for S5 (see (Halpern and Rêgo 2007) for more details). Since SAT solvers became a quite efficient practical NP-oracle for many problems, we are interested in studying SAT encoding for the S5-SAT problem from a practical perspective. Using a SAT-oracle in the context of modal logic is not new: a comprehensive overview of previous works can be found for example in (Sebastiani and Tacchella 2009). However, most of them tackle the satisfiability of modal logic K. *SAT (Giunchiglia, Giunchiglia, and Tacchella 1999; Giunchiglia and Tacchella 2000; Giunchiglia et al. 2000; Giunchiglia, Tacchella, and Giunchiglia 2002) uses a SAT-oracle to decide the satisfiability of 8 different modal logics, including K, but not S5. In the same spirit as our work, a translation of modal logic K to SAT has been proposed in Km2SAT (Giunchiglia and Sebastiani 2000; Sebastiani and Vescovi 2009). More recently, the solver InKreSAT (Kaminski and Tebbi 2013) proposed an innovative SAT-based system where the SAT solver drives the development of a tableaux method. Theoretically, an approach based on Satisfiability Modulo Theory (Areces, Fontaine, and Merz 2015) has also been proposed. None of those methods are applicable to modal logic S5.

The number of variables required to reduce S5-SAT to SAT depends on an upper bound of the number of possible worlds to be considered in the S5-model. Minimizing that upper bound is thus crucial to obtain CNF formulas of reasonable size. We propose a new upper bound based on a syntactical property of the formula, that we call "diamond degree". We provide some experimental evidences that our approach improves significantly the state-of-the-art S5 solvers.

The remainder of the paper is organized as follows: we first present the modal logic S5 and the different bounds on the size of a S5-models; then we detail the reduction from S5-SAT to SAT, parameterized by the number of worlds to consider. We present two improvements to reduce the size of the SAT encoding: a better upper bound on the number of worlds to consider and structural caching. Finally, we compare experimentally the efficiency of our approach to state-of-the-art S5 solvers.

Preliminaries

Let $\mathbb{P}$ be a finite non-empty set of propositional variables. The language $\mathcal{L}$ of the modal logic S5 is the set of formulas ϕ defined by the following grammar in BNF:

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \Box\phi \mid \Diamond\phi$$

where p ranges over $\mathbb{P}$. A formula of the form $\Box\phi$ (*box phi*) means ϕ is necessarily true. A formula of the form $\Diamond\phi$ (*di-*

amond phi) means ϕ is possibly true. Any formula $\phi \in \mathcal{L}$ may be translated to an equivalent formula in negation normal form (NNF), that is, negations appear only in front of propositional variables (see the definition in (Robinson and Voronkov 2001)), noted $\mathrm{nnf}(\phi)$. This can be done in polynomial time. Formulas in $\mathcal{L}$ are interpreted using pointed S5-models. A S5-model is a pair (W, V), where W is a non-empty set of possible worlds and V is a valuation function with signature $W \to (\mathbb{P} \to \{0, 1\})$. A pointed S5-model is a triplet (W, V, w), where (W, V) is a S5-model and $w \in W$. The satisfaction relation $\models$ between formulas in $\mathcal{L}$ and pointed S5-models is recursively defined as follows:

$(W, V, w) \models \top$

$(W, V, w) \models p$ iff $V(w)(p) = 1$

$(W, V, w) \models \neg\phi$ iff $(W, V, w) \not\models \phi$

$(W, V, w) \models \phi_1 \wedge \phi_2$ iff $(W, V, w) \models \phi_1$ and $(W, V, w) \models \phi_2$

$(W, V, w) \models \phi_1 \vee \phi_2$ iff $(W, V, w) \models \phi_1$ or $(W, V, w) \models \phi_2$

$(W, V, w) \models \Box\phi$ iff for all $w' \in W$ we have $(W, V, w') \models \phi$

$(W, V, w) \models \Diamond\phi$ iff there is $w' \in W$ s.t. $(W, V, w') \models \phi$

Validity and satisfiability are defined as usual. A formula $\phi \in \mathcal{L}$ is valid, noted $\models \phi$, if and only if, for all pointed S5-models (W, V, w), we have $(W, V, w) \models \phi$. Moreover, ϕ is satisfiable if and only if $\not\models \neg\phi$.

From S5-SAT to SAT

It was shown in (Ladner 1977) that if an S5 formula ϕ with n modal connectives is satisfiable, then there is an S5-model satisfying ϕ with at most $n + 1$ worlds. As a consequence, we know that there exists an algorithm running in polynomial time able to transform the S5-SAT problem into the SAT problem. However, to the best of our knowledge, no one evaluated this approach in practice. It has not been compared against state-of-the-art solvers until now. Yet SAT-based approaches are known to be quite efficient in practice.

A SAT encoding is represented here by a translation function tr, which takes as input an S5-formula ϕ and a number of worlds n in the S5-model and produces a propositional formula. This is inspired by the standard translation to FOL (Patrick Blackburn and Wolter 2007). Note that the accessibility relation does not need to be represented in an S5-model, because it is an equivalence relation.

$$\mathrm{tr}(\phi, n) = \mathrm{tr}'(\mathrm{nnf}(\phi), 1, n)$$

$$\mathrm{tr}'(\top, i, n) = \top \qquad \mathrm{tr}'(\neg\top, i, n) = \neg\top$$

$$\mathrm{tr}'(p, i, n) = p_i \qquad \mathrm{tr}'(\neg p, i, n) = \neg p_i$$

$$\mathrm{tr}'((\phi \wedge \cdots \wedge \delta), i, n) = \mathrm{tr}'(\phi, i, n) \wedge \cdots \wedge \mathrm{tr}'(\delta, i, n)$$

$$\mathrm{tr}'((\phi \vee \cdots \vee \delta), i, n) = \mathrm{tr}'(\phi, i, n) \vee \cdots \vee \mathrm{tr}'(\delta, i, n)$$

$$\mathrm{tr}'(\Box\phi, i, n) = \bigwedge_{j=1}^{n}(\mathrm{tr}'(\phi, j, n))$$

$$\mathrm{tr}'(\Diamond\phi, i, n) = \bigvee_{j=1}^{n}(\mathrm{tr}'(\phi, j, n))$$

The translation adds fresh Boolean variables p_i to the formula, denoting the truth value of p in the world w_i. In such function, the i parameter represents the index of the world. The function is defined over a Negative Normal Form (NNF) formula for sake of simplicity.

Example 1. *Let $\phi = \Diamond(a \wedge \Box b)$, Figure 1 shows the effect of applying* tr *to the formula ϕ with $n = 2$.*

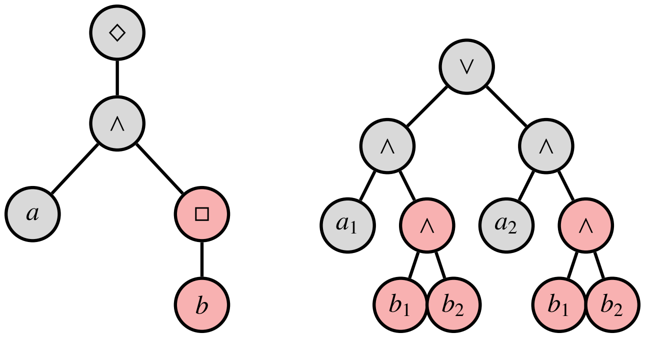


Figure 1: From S5 ϕ (left) to propositional logic with n=2 (right).

If the value of n is an upper-bound of the number of worlds in the model, then the translation is equi-satisfiable to the original S5 formula. In particular, it is the case for the upper-bound shown in (Ladner 1977):

Definition 1. *Let ϕ be in $\mathcal{L}$, $\mathtt{nm}(\phi)$ denotes the number of modal connectives in the formula ϕ.*

Theorem 1. *$\phi \in \mathcal{L}$ is satisfiable if and only if $tr(\phi, \mathtt{nm}(\phi)+1)$ is satisfiable.*

Proof. Theorem 1 is proved in the same way as the standard translation to FOL plus Lemma 6.1 in (Ladner 1977). □

Note that the result of the translation is not in CNF. As such, classical translation into CNF such as (Tseitin 1983) is needed to use a SAT oracle.

Improvements on the upper-bound

The size of the encoding depends on $\mathtt{nm}(\phi)$. In practice, such upper bound produces unreasonably large formulas. As such, a first step is to improve that upper bound. We propose to use a new metric called the diamond degree as a new upper bound.

Definition 2 (Diamond-Degree)**.** *The diamond degree of $\phi \in \mathcal{L}$, noted $dd(\phi)$, is defined recursively, as follows:*

$$dd(\phi) = dd'(\mathrm{nnf}(\phi))$$

$$dd'(\top) = dd'(\neg\top) = dd'(p) = dd'(\neg p) = 0$$

$$dd'(\phi \wedge \psi) = dd'(\phi) + dd'(\psi)$$

$$dd'(\phi \vee \psi) = \max(dd'(\phi), dd'(\psi))$$

$$dd'(\Box\phi) = dd'(\phi) \qquad dd'(\Diamond\phi) = 1 + dd'(\phi)$$

The computation of the diamond degree requires to convert ϕ in NNF. As mentioned in the previous section, this operation can always be performed in both polynomial time and space. Then, the diamond degree of any formula can be computed in polynomial time. Without loss of generality, we will assume that $\phi \in$ NNF and consider dd' instead of dd.

Informally, the diamond degree represents an upper bound of the number of diamonds to be taken into account to satisfy the formula. To show that the diamond degree is a valid upper bound for our SAT encoding, we will use a tableau method. Therefore, we need some additional definitions. Let ϕ be a formula in NNF and let sub(ϕ) be the set of all sub-formulas of ϕ. A tableau for ϕ is a non-empty set $T = \{s_0, s_1, \ldots, s_n\}$ such that each $s_i \in T$ is a subset of sub(ϕ) and $\phi \in s_0$. In addition, each set $s_i \in T$ satisfies the following conditions:

1. $\neg\top \notin s$.
2. if $p \in s$ then $\neg p \notin s$.
3. if $\neg p \in s$ then $p \notin s$.
4. if $\psi_1 \wedge \psi_2 \in s$ then $\psi_1 \in s$ and $\psi_2 \in s$.
5. if $\psi_1 \vee \psi_2 \in s$ then $\psi_1 \in s$ or $\psi_2 \in s$.
6. if $\Box\psi_1 \in s$ then $\forall s' \in T$ we have $\psi_1 \in s'$.
7. if $\Diamond\psi_1 \in s$ then $\exists s' \in T$ s.t. $\psi_1 \in s$.

Lemma 1. *Let ϕ be in NNF. There is a tableau for ϕ if and only if ϕ is satisfiable.*

Proof. Direct from the definition of the NNF (which preserves the satisfiability) and from the definition of the tableau method (there will be a tableau for ϕ if and only if ϕ is satisfiable). $\Box$

Lemma 2. *Let ϕ be in NNF. The number of elements of the set T created by constructing its tableau is bounded by* $\mathsf{dd}'(\phi) + 1$.

Proof. Let $\psi \in$ sub(ϕ). Let $g(\psi)$ be the number of sets s added to T because of ψ. That is, $g(\psi)$ is the number of times the condition involving operator $\Diamond$ is triggered for sub-formulas of ψ. We show that, for all $\psi \in$ sub(ϕ) we have $g(\psi) \leq \mathsf{dd}'(\psi)$. We do so by induction on the structure of ψ.

Induction base. We consider four cases: (1) $\psi = \top$, (2) $\psi = \neg\top$, (3) $\psi = p$ and (4) $\psi = \neg p$. In all cases, the condition involving $\Diamond$ will never be triggered for formulas in sub(ψ). Then $g(\psi) = 0 \leq \mathsf{dd}'(\psi)$.

Induction step. We consider four cases:

1. $\psi = \psi_1 \wedge \psi_2$. Assume $\psi \in s$, for some $s \in T$. In this case, the algorithm adds ψ_1 and ψ_2 to s. Therefore, $g(\psi)$ is bounded by $g(\psi_1) + g(\psi_2)$. The latter is bounded by $\mathsf{dd}'(\psi_1) + \mathsf{dd}'(\psi_2)$ (by the induction hypothesis). Then $g(\psi) \leq \mathsf{dd}'(\psi)$.
2. $\psi = \psi_1 \vee \psi_2$. Assume $\psi \in s$, for some $s \in T$. In this case, the algorithm adds either ψ_1 or ψ_2 to s. Therefore, $g(\phi)$ is bounded by $\max(g(\psi_1), g(\psi_2))$. The latter is bounded by $\max(\mathsf{dd}'(\psi_1), \mathsf{dd}'(\psi_2))$ (by the induction hypothesis). Then $g(\psi) \leq \mathsf{dd}'(\psi)$.
3. $\psi = \Box\psi_1$. Assume $\psi \in s$, for some $s \in T$. In this case, the algorithm adds ψ_1 to all $s' \in T$. Therefore, $g(\psi)$ is bounded by $g(\psi_1)$. The latter is bounded by $\mathsf{dd}'(\psi_1)$ (by the induction hypothesis). Then $g(\psi) \leq \mathsf{dd}'(\psi)$.
4. $\psi = \Diamond\psi_1$. Assume $\psi \in s$, for some $s \in T$. In this case, if there is no s' containing ψ_1 then the algorithm adds a new s'' to T and adds ψ_1 to s''. Therefore, $g(\psi)$ is bounded by $1 + g(\psi_1)$. The latter is bounded by $1 + \mathsf{dd}'(\psi_1)$ (by the induction hypothesis). Then $g(\psi) \leq \mathsf{dd}'(\psi)$.

Therefore, we have $|T| = 1 + g(\phi) \leq 1 + \mathsf{dd}'(\phi)$. $\Box$

Thus, for any $\phi \in \mathcal{L}$, each s_i of the tableau T corresponds to a $w_i \in W$ in the S5-model, $|T| \leq \mathsf{dd}(\phi) + 1$ means that the number of worlds in the S5-model is bounded by $\mathsf{dd}(\phi) + 1$.

Theorem 2. *$\phi \in \mathcal{L}$ is satisfiable if and only if $tr(\phi, \mathsf{dd}(\phi)+1)$ is satisfiable.*

Structural Caching

Caching is a classical way to avoid redundant work. *SAT performs caching using a "bit matrix" (Giunchiglia and Tacchella 2001). Efficient implementation of BDD (Bryant 1986) packages also rely in caching, to build an explicit graph. These two examples require additional time and space to search and cache already performed works. Here, our technique is a "simple but efficient" trade-off. It does not memoize the work, so it may not cache all possible formulas, but it only requires a flag to detect redundant work.

In the example depicted in Figure 2.b, sub-formula ($b_1 \wedge b_2$) appears twice. The translation of the first diamond creates two sub-formulas $a_1 \wedge \Diamond b$ and $a_2 \wedge \Diamond b$, where each $\Diamond b$ needs to be translated.

Because we are in S5 (all worlds are connected), the translations of $\Diamond b$ on different worlds are equivalent, so we can reuse the same sub-formula. It means that instead of using a tree, we can work with a DAG, which allows a more efficient translation to CNF.

Lemma 3. $\mathrm{tr}'(\circ\phi, i, n) = \mathrm{tr}'(\circ\phi, j, n)\ \forall i, j$ *and* $\circ \in \{\Diamond, \Box\}$

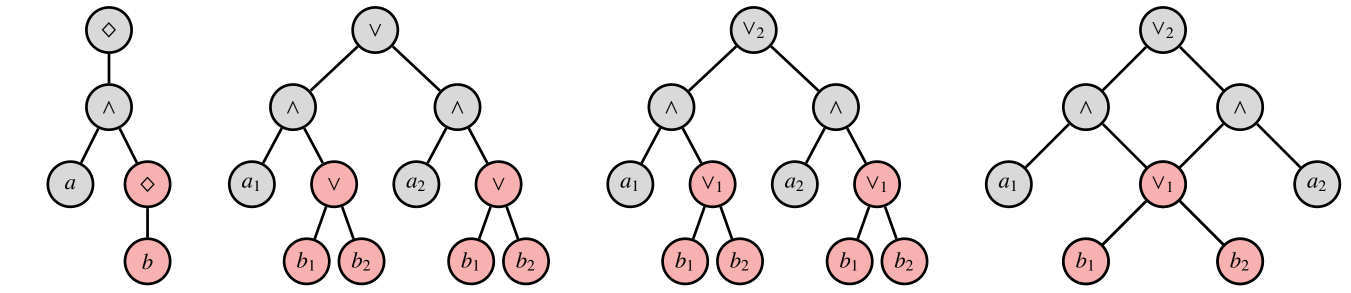
Proof of Lemma 3. By definition, there are only 2 cases:
- ($\circ = \Box$) then, $\mathrm{tr}'(\Box\phi, i, n) = \bigwedge_{k=1}^{n}(\mathrm{tr}'(\phi, k, n))$
- ($\circ = \Diamond$) then $\mathrm{tr}'(\Diamond\phi, i, n) = \bigvee_{k=1}^{n}(\mathrm{tr}'(\phi, k, n))$

In each case, the result is independent from i, so choosing j as an index gives the exact same result. $\Box$

Informally, lemma 3 shows the fact that no matter how embedded the modal sub-formula is, its translation will always give the same result (independent from the index i). Therefore, we can start by translating the most embedded sub-formula, tag the corresponding node and backtrack. The resulting formula may contain several nodes with the same tag. This means that these sub-formulas are syntactically identical (see Figure 2.c). Then, we maintain only one occurrence of the sub-formula, transforming the tree in a DAG (see Figure 2.d). Structural caching is thus performed on the fly before translating to CNF. The translation function using this technique is noted tr^{+}.

Experiments

We considered LCKS5TabProver (Abate, Goré, and Widmann 2007) and SPASS 3.7 (Weidenbach et al. 2009), which are, to the best of our knowledge, the state-of-the-art in



(a) $\phi = \Diamond(a \wedge \Diamond b)$ (b) $\mathrm{tr}(\phi, \mathrm{dd}(\phi))$ (c) $\mathrm{tr}^{+}(\phi, \mathrm{dd}(\phi))$ (d) $\mathrm{tr}^{+}(\phi, \mathrm{dd}(\phi))$ with simplification

Figure 2: Translation of $\Diamond(a \wedge \Diamond b)$ (left), initial translation (middle-left), tagged formula (middle-right), final translation (right)

S5-satisfiability solving. We compared them to 4 different configurations of S52SAT: `nm`, `nm+` (with caching), `dd`, `dd+` (with caching) (http://www.cril.univ-artois.fr/ montmirail/s52SAT). In order to evaluate rigorously the solvers, we used the same input for each solver, in InToHyLo format (Hoffmann 2010). For this purpose, we modified the code of LCKS5TabProver to make it able to read that format. The modifications are simple enough to guarantee that the performance of the solver is not affected.

We used SPASS 3.7 and not the latest available version 3.9 because the former reads `dfg` format which can be produced from InToHyLo benchmarks using the tool `ftt` (Fast Transformation Tool) embedded in Spartacus (Götzmann, Kaminski, and Smolka 2010). The difference between 3.7 and 3.9 is minimal according to the solver's web site. The translation time is negligible in our experiments. We use Glucose 4.0 (Audemard and Simon 2009) as backend SAT solver. We also considered MetTel2 (Tishkovsky, Schmidt, and Khodadadi 2012) and LoTREC (Gasquet et al. 2005) but unfortunately they are not designed to efficiently solve modal logic S5 problems. We evaluated these solvers on well established modal logic benchmarks: $3CNF_K$ (Patel-Schneider and Sebastiani 2003), $MQBF_K$ (Massacci 1999), $TANCS2000_K$ (Massacci and Donini 2000) and $LWB_{K,KT,S4}$ (Balsiger, Heuerding, and Schwendimann 2000). Note that they are designed for modal logic K, KT and S4. As a consequence, some of them are trivial in the S5 setting. However, we believe that the results on those benchmarks are still significant. S5-SAT entails K, KT and S4-SAT, so we could envision S5-SAT as a preprocessing step for those modal logics.

We set the memory limit to 8GB and the runtime limit to 900 seconds. We rarely reached the timeout (804 times out of 12444 runs). Most unsolved benchmarks are due to lack of memory.

In the following tables, we provide the number of benchmarks solved, in bold the best results of a given row/column (according to the orientation of the Table); and between parenthesis, we provide the number of benchmarks which cannot be solved because of lack of memory, if any.

3CNF_K : Caching does not help

The results are displayed in the Table 1. `dd`, `dd+` and SPASS are quite close to each other. The formulas consist of big conjunctions where each conjunct has modal depth of at most 2,4 or 6. They are constructed in such a way that our caching algorithm could not find redundancies on such formulas. This is why the caching does not provide any benefit on those benchmarks.

Solver	d=2	d=4	d=6	Total
LckS5TabProver	0 (17)	0 (29)	0 (40)	0
S52SAT `nm`	21 (24)	0 (45)	0 (45)	21
S52SAT `nm+`	21 (24)	0 (45)	0 (45)	21
S52SAT `dd`	**45 (0)**	**8 (27)**	0 (45)	**53**
S52SAT `dd+`	**45 (0)**	**8 (27)**	0 (45)	**53**
SPASS 3.7	**45 (0)**	5 (40)	0 (45)	50

Table 1: #instances solved in $3CNF_K$

modKSSS and modKLadn : Caching helps

n,*a*	#	LckS5...	`nm`	`nm+`	`dd`	`dd+`	SPASS 3.7
4,4	40	0 (12)	32	**40**	**40**	**40**	**40**
4,6	40	0 (23)	32	**40**	**40**	**40**	32 (8)
8,4	40	0 (16)	32	**40**	39 (1)	**40**	16 (24)
8,6	40	0 (17)	24	**40**	**40**	**40**	10 (30)
16,4	40	0 (10)	22	**40**	**40**	**40**	8 (32)
16,6	40	0 (16)	19	**40**	39 (1)	**40**	2 (38)
total	240	0	161	**240**	238	**240**	108
4,4	40	**40**	0 (40)	**40**	0 (40)	**40**	0 (40)
4,6	40	14 (0)	0 (40)	32 (8)	0 (40)	**40**	0 (40)
8,4	40	2 (0)	0 (40)	8 (32)	0 (40)	**40**	0 (40)
8,6	40	0 (0)	0 (40)	0 (40)	0 (40)	**8** (32)	0 (40)
16,4	40	0 (0)	0 (40)	0 (40)	0 (40)	0 (40)	0 (40)
16,6	40	0 (0)	0 (40)	0 (40)	0 (40)	0 (40)	0 (40)
total	240	56	0	80	0	**128**	0

Table 2: Upper: modKSSS — Lower: modKLadn

n represents the number of variables and *a* represents the number of alternations in the original QBF prefix, see (Massacci 1999) for more details, and # corresponds to the number of benchmarks available for a given pair (*n*,*a*).

In this category, we can see that `dd` and `dd+` are much more efficient than SPASS. The formulas in these benchmarks contain a lot of redundancies: it can be seen that enabling caching allows us to solve all modKSSS benchmarks for instance.

TANCS-2000 : Memory Demanding

n,*a*	#	LckS5...	nm	nm+	dd	dd+	SPASS 3.7
4,4	40	0 (38)	**40**	**40**	**40**	**40**	**40**
4,6	40	0 (33)	**40**	**40**	**40**	**40**	**40**
8,4	40	0 (38)	**40**	**40**	**40**	**40**	**40**
8,6	40	0 (39)	**40**	**40**	**40**	**40**	**40**
16,4	40	0 (40)	**40**	**40**	**40**	**40**	**40**
16,6	40	0 (40)	**40**	**40**	**40**	**40**	35 (5)
total	240	0	**240**	**240**	**240**	**240**	235
4,4	40	23 (0)	3 (37)	**40**	**40**	**40**	**40**
4,6	40	3 (0)	0 (40)	**40**	**40**	**40**	31 (9)
total	80	26	3	**80**	**80**	**80**	71

Table 3: TANCS-2000. Upper: qbfMS — Lower: qbfML

Here, we can see that these problems can be solved by almost all the solvers. The instances not solved by SPASS 3.7 are due to lack of memory. As pointed out in the following section, by increasing the memory limit to 32GB, these instances are solved by SPASS.

LWB K, KT, S4 : Both dd and caching help

Solver	Logic K	Logic KT	Logic S4	Total
LckS5TabProver	227 (73)	206 (102)	194 (111)	627
S52SAT nm	307 (66)	344 (33)	292 (86)	943
S52SAT nm+	351 (21)	363 (12)	349 (28)	1063
S52SAT dd	333 (40)	355 (23)	337 (40)	1025
S52SAT dd+	**357 (12)**	**364 (12)**	**363 (12)**	**1084**
SPASS 3.7	343 (16)	363 (15)	360 (17)	1066

Table 4: #instances solved in $LWB_{K,KT,S4}$

The benchmarks are originally separated on SAT/UNSAT formulas for the logics K, KT and S4. Obviously, the satisfiability in S5 may differ so we removed the SAT/UNSAT separation. The results are displayed in the Table 4. Here again, dd+ performs slightly better than SPASS. The benchmarks which cannot be solved are a specific modal logic encoding of the pigeon hole principle (Haken 1985) in the corresponding logic.

Overall results on all the benchmarks

Solver	# solved	# SAT	MO	TO
LckS5TabProver	709	143	710	655
S52SAT nm	1377	411	667	30
S52SAT nm+	1733	452	292	49
S52SAT dd	1645	433	412	17
S52SAT dd+	**1834**	**460**	**203**	37
SPASS 3.7	1530	451	528	**16**

Table 5: Overall results on all the benchmarks

SPASS is outperformed in part because of its poor results on the modKSSS and modKLadn benchmarks. While the default SAT encoding often exhausts the available memory, each of the two proposed improvements significantly reduces the number of memory-out, the best results being obtained when both are enabled.

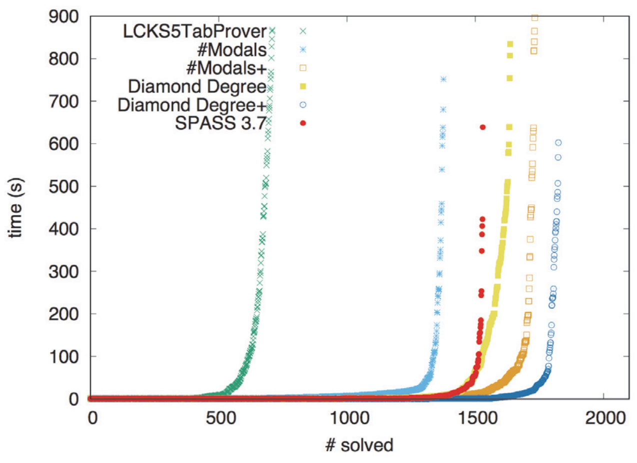


Figure 3: runtime distribution

It seems pretty clear that the structural caching is key in our approach to efficiently solve those benchmarks. This is witnessed by the scatter plot in Figure 4. The x-axis corresponds to the time used by dd and the y-axis corresponds to the time used by dd+ to solve the problem.

Solver	avg	median	max
nm	6 881 821	1 100 054	58 653 264
nm+	1 619 923	118 040	29 492 779
dd	2 385 515	169 324	55 813 557
dd+	269 891	27 090	22 914 442

Table 6: Number of clauses in the generated CNF formulas

The main reason of the improvement is the reduction of the size of the CNF encoding, as shown by Table 6. We had a median value of 1,100,054 clauses for nm and this value drops down to 27,090 for dd+ on the exact same problems.

Importance of the memory

We repeated the experiments with 32GB of RAM. Note that for the SAT based approach, the lack of memory happens during the translation phase, while for the others, the memory is exhausted in the solving phase.

One may wonder what would be the performance with more memory. Table 7 summarizes the number of problems solved with 8GB and 32GB by each solver. Providing 32GB to SPASS does not change the results significantly: it solved 30 additional instances. Our SAT approach also benefits from that increased amount of memory, up to 98 additional benchmarks can be solved by nm without caching. 32 GB is sufficient for LckS5TabProver (no memory-out), however only one additional benchmark can be solved.

Figure 6 compares the memory consumption of SPASS 3.7 and S52SAT dd+ in MB. SPASS usually requires more memory than dd+. The runtime and the memory consumption are roughly correlated (the Pearson product-moment

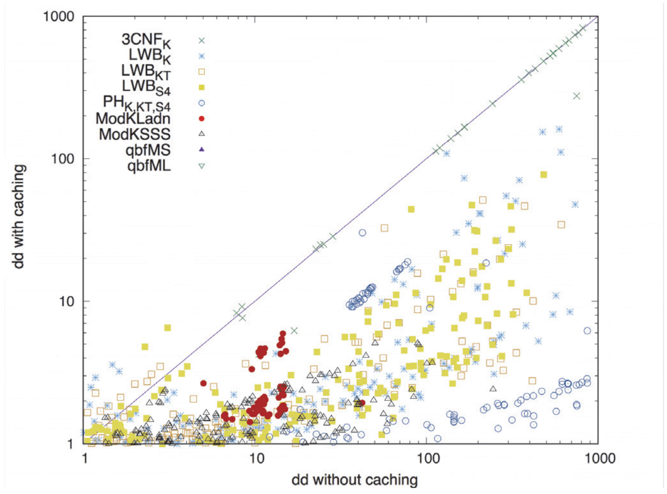


Figure 4: Runtime of dd with and without caching

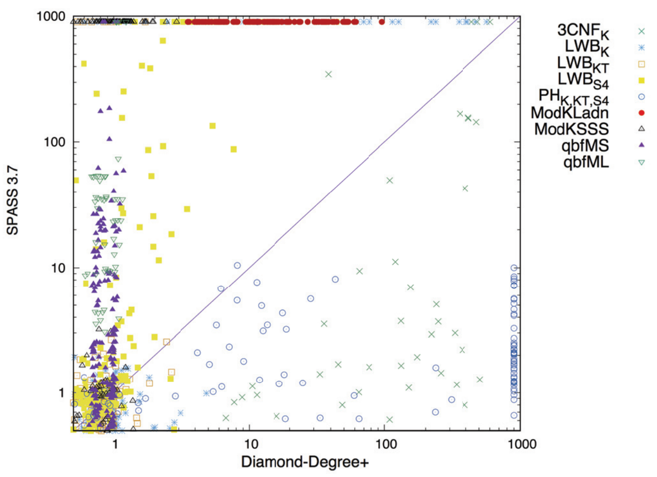


Figure 5: Runtime of SPASS 3.7 vs dd+

Solver	8GB	32GB	MO with 32GB
LckS5TabProver	709	710	0
SPASS 3.7	1530	1560	498
S52SAT `nm`	1377	1475	382
S52SAT `nm+`	1733	1800	166
S52SAT `dd`	1645	1672	317
S52SAT `dd+`	1834	1888	96

Table 7: #Solved with 8GB and 32GB

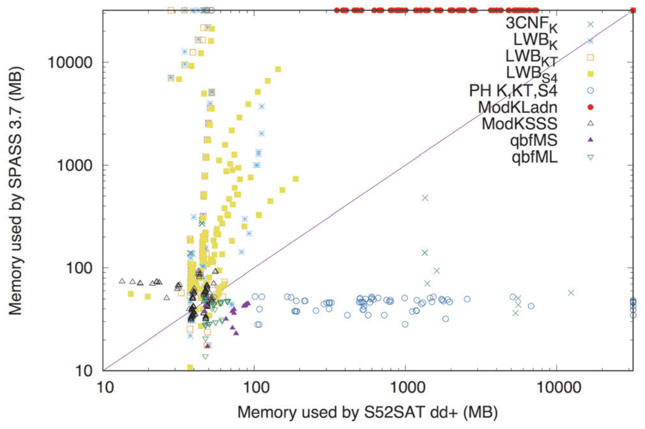


Figure 6: Memory comparison (32 MB) SPASS 3.7 vs dd+

correlation coefficient for dd+ is equal to 0.58 and for SPASS it is 0.57). In the category ModKLadn, SPASS runs out of memory even with 32 GB.

Comparison against the state-of-the-art

Figure 5 shows a detailed runtime comparison between dd+ and SPASS on all benchmarks. In most cases, our approach outperforms SPASS. The two cases where it is less efficient consists in the hard combinatorial UNSAT Pigeon-Hole problems and the so-called "3CNF" benchmarks on which neither the diamond degree nor the caching helps.

Conclusion

We presented a new SAT encoding to solve the S5-SAT problem using a SAT solver. It is based on a reduction from S5-SAT to SAT with two improvements: a better upper bound on the number of required worlds and a structural caching. We compared our approach against solvers representing, to the best of our knowledge, the state-of-the-art for practical S5-SAT solving, on a wide range of classical modal logic benchmarks. The SAT based approach with all improvements enabled outperformed those solvers.

Even if the benchmarks may not be representative of practical S5 benchmarks because they come from other modal logics (K, KT, S4), those results open interesting perspectives. It is indeed the case that proving the satisfiability of a modal logic formula in S5 entails that the formula is also satisfiable on less restrictive systems (i.e. in K, KT, S4). Since our S5-solver provides a S5-model in just a few seconds (2.06s median time), we could perfectly use it as a preprocessing step to solve benchmarks in other modal logics.

Preliminary results on that direction are quite encouraging: on 276 satisfiable benchmarks in logic S5 (thus in K), S52SAT outperforms the state of the art Spartacus (Götzmann, Kaminski, and Smolka 2010) on 158 of them. Another perspective, would be to adapt S52SAT in order to solve KD45-SAT problems given the fact that KD45 is also NP-Complete (Fagin et al. 1995).

Acknowledgments

We thank the anonymous reviewers for their insightful comments. We thank Renate Schmidt for providing help on finding S5 solvers. Part of this work was supported by the French Ministry for Higher Education and Research and the Nord-Pas de Calais Regional Council through the "Contrat de Plan État Région (CPER)" and by an EC FEDER grant.

References

Abate, P.; Goré, R.; and Widmann, F. 2007. Cut-free single-pass tableaux for the logic of common knowledge. In *Workshop on Agents and Deduction at TABLEAUX'07*.

Areces, C.; Fontaine, P.; and Merz, S. 2015. Modal satisfiability via SMT solving. In *Software, Services, and Systems - Essays Dedicated to Martin Wirsing*, 30–45.

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In *Proc. of IJCAI'09*, 399–404.

Balsiger, P.; Heuerding, A.; and Schwendimann, S. 2000. A Benchmark Method for the Propositional Modal Logics K, KT, S4. *J. Autom. Reasoning* 24(3):297–317.

Bienvenu, M.; Fargier, H.; and Marquis, P. 2010. Knowledge compilation in the modal logic S5. In *Proc. of AAAI'10*.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers* 35(8):677–691.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning About Knowledge*. The MIT Press.

Fairtlough, M., and Mendler, M. 1994. An Intuitionistic Modal Logic with Applications to the Formal Verification of Hardware. In *Proc. of CSL'94*, 354–368.

Fitting, M. 2000. Modality and databases. In *Proc. of TABLEAUX'00*, 19–39.

Gasquet, O.; Herzig, A.; Longin, D.; and Sahade, M. 2005. LoTREC: Logical Tableaux Research Engineering Companion. In *Proc. of TABLEAUX'05*, 318–322.

Giunchiglia, F., and Sebastiani, R. 2000. Building Decision Procedures for Modal Logics from Propositional Decision Procedures: The Case Study of Modal K(m). *Inf. Comput.* 162(1-2):158–178.

Giunchiglia, E., and Tacchella, A. 2000. System description: *SAT: A platform for the development of modal decision procedures. In *Proc. of CADE'00*, 291–296. Springer.

Giunchiglia, E., and Tacchella, A. 2001. A subset-matching size-bounded cache for testing satisfiability in modal logics. *Ann. Math. Artif. Intell.* 33(1):39–67.

Giunchiglia, E.; Sebastiani, R.; Giunchiglia, F.; and Tacchella, A. 2000. SAT vs. Translation based decision procedures for modal logics: a comparative evaluation. *Journal of Applied Non-Classical Logics* 10(2):145–172.

Giunchiglia, E.; Giunchiglia, F.; and Tacchella, A. 1999. The sat-based approach for classical modal logics. In *Proc. of AI*IA'99*, 95–106.

Giunchiglia, E.; Tacchella, A.; and Giunchiglia, F. 2002. SAT-Based Decision Procedures for Classical Modal Logics. *J. Autom. Reasoning* 28(2):143–171.

Götzmann, D.; Kaminski, M.; and Smolka, G. 2010. Spartacus: A tableau prover for hybrid logic. *Electr. Notes Theor. Comput. Sci.* 262:127–139.

Haken, A. 1985. The intractability of resolution. *Theoretical Computer Science* 39(0):297 – 308.

Halpern, J. Y., and Rêgo, L. C. 2007. Characterizing the NP-PSPACE Gap in the Satisfiability Problem for Modal Logic. In *Proc. of IJCAI'07*, 2306–2311.

Hoffmann, G. 2010. *Tâches de raisonnement en logiques hybrides*. Ph.D. Dissertation, Henri Poincaré University, Nancy.

Kaminski, M., and Tebbi, T. 2013. InKreSAT: Modal Reasoning via Incremental Reduction to SAT. In *Proc. of CADE'13*, 436–442.

Ladner, R. E. 1977. The Computational Complexity of Provability in Systems of Modal Propositional Logic. *SIAM J. Comput.* 6(3):467–480.

Lorini, E., and Schwarzentruber, F. 2010. A modal logic of epistemic games. *Games* 1(4):478–526.

Massacci, F., and Donini, F. M. 2000. Design and results of TANCS-2000 non-classical (modal) systems comparison. In *Proc. of Tableaux'00*, 52–56.

Massacci, F. 1999. Design and results of the tableaux-99 non-classical (modal) systems comparison. In *Proc. of Tableaux'99*, 14–18.

Masssacci, F. 2000. Single step tableaux for modal logics: Methodology, computations, algorithms. *Journal of Autom.* 24(3):319–364.

Niveau, A., and Zanuttini, B. 2016. Efficient Representations for the Modal Logic S5. In *Proc. of IJCAI'16*, 1223–1229.

Patel-Schneider, P. F., and Sebastiani, R. 2003. A new general method to generate random modal formulae for testing decision procedures. *J. Artif. Intell. Res.* 18:351–389.

Patrick Blackburn, J. v. B., and Wolter, F. 2007. *Handbook of Modal Logic*. Elsevier.

Robinson, J. A., and Voronkov, A., eds. 2001. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press.

Sebastiani, R., and Tacchella, A. 2009. SAT Techniques for Modal and Description Logics. *Handbook of Satisfiability* 185:781–824.

Sebastiani, R., and Vescovi, M. 2009. Automated Reasoning in Modal and Description Logics via SAT Encoding: the Case Study of K(m)/ALC-Satisfiability. *JAIR* 35(1):343.

Sebastiani, R., and Villafiorita, A. 1998. Sat-based decision procedures for normal modal logics: A theoretical framework. In *Proc. of AIMSA'98*, 377–388.

Tishkovsky, D.; Schmidt, R. A.; and Khodadadi, M. 2012. The Tableau Prover Generator MetTeL2. In *Proc. of JELIA'12*, 492–495.

Tseitin, G. S. 1983. *On the Complexity of Derivation in Propositional Calculus*. Springer. 466–483.

VII, T. M.; Crary, K.; and Harper, R. 2005. Distributed control flow with classical modal logic. In *Proc. of CSL'05*, 51–69.

Weidenbach, C.; Dimova, D.; Fietzke, A.; Kumar, R.; Suda, M.; and Wischnewski, P. 2009. SPASS version 3.5. In *Proc. of CADE'09*, 140–145.