Structured Problems for Modal Satisfiability Testing* (Preliminary Report)

Juan Heguiabehere Gabriel Infante López Maarten de Rijke

Institute for Logic, Language and Computation (ILLC)
Faculty of Science, University of Amsterdam
Nieuwe Achtergracht 166, 1018 WV Amsterdam, The Netherlands
E-mail: \{juanh,infante,mdr\}@science.uva.nl

Experimental Paper

Abstract

We report on ongoing experimental work on evaluating test sets for testing modal satisfiability solvers. Our longterm aim is to understand the difference between the use of structured and of unstructured randomly generated problems. Which parts of the problem space do they explore?

1 Introduction

Usually, theoretical studies do not provide an indication of the effectiveness and behavior of complex systems such as satisfiability solvers. Instead, empirical evaluations have to be used. In the area of propositional satisfiability checking there is large and rapidly expanding body of experimental knowledge; see e.g., [3]. In contrast, empirical aspects of modal satisfiability checking have only recently drawn the attention of researchers. We now have a number of test sets, some of which have been evaluated extensively [2, 7, 4, 9, 8]. In addition, we also have a clear set of guidelines for performing empirical testing in the setting of modal logic [8, 7].

Currently, there are two main test methodologies for modal satisfiability solvers, both based on randomly generating problems. The first random generation technique used in testing modal decision procedures, the random 3CNF$_{\Box_m}$ test methodology, was proposed in [4]; its subsequent development is described in [8], and its most recent incarnation is given Patel-Schneider and Sebastiani [14]. It allows one to explore the geography of modal satisfiability problems in as broad a manner as possible. A potential problem with this generator is that we have no idea whether the randonly generated formulas are representative of formulas generated by realistic applications.

There is some hope that the latter point is addressed by the other main test methodology that is currently in use for evaluating modal provers: the random modal quantified boolean formula (QBF) test set. Quantified boolean formulas are interesting for a number of reasons. Many real-world problems can naturally be rephrased in terms of QBFs [15], and, hence, the performance of modal provers on QBFs is a good indication of their performance on those real-world problems. Moreover, the QBF-validity problem (the problem of deciding whether a QBF without free variables is valid) is known to be PSPACE-complete; since most of the logics that we're interested have satisfiability problems that are (at least) PSPACE hard, in principle QBFs provide problems of the appropriate complexity.

*Juan Heguiabehere and Maarten de Rijke were supported by the Spinoza project 'Logic in Action.' Gabriel Infante López and Maarten de Rijke were supported by a grant from the Netherlands Organization for Scientific Research (NWO) under project number 220-80-001. Maarten de Rijke was also supported by grants from NWO under project numbers 612-13-001, 365-20-005, 612.069.006, and 612.000.106. We would like to thank our anonymous referees for valuable comments.

The random modal QBF test set was proposed by Massacci [11], and used in the 1999 and 2000 editions of the TANCS system comparisons [18]. The random modal QBF tests are performed on a single data point, and the results are presented in the form of tables, each entry consisting of the number of successful solutions and the mean CPU time for such solutions. Systems are compared by their numbers of successful solutions, and, if that results in a tie, by their mean CPU times.

In this paper we're interested in a different use of the random modal QBF test set, viz. in its use for evaluating the qualitative and quantitative behavior of modal satisfiability solvers, and hence we're interested in plots rather than tables. More specifically, our longterm aim with this work is to understand what can and cannot learn from structured test sets, such as the modal QBF test set, and how it compares to using randomly generated unstructured problems such as the ones generated by the CNF methodology described above.

We're still a long way from having final answers to these questions. In this paper we review what's currently known about the random modal QBF test set, and discuss a number of concrete issues that are guiding our ongoing experiments. We start by recalling the ideas underlying the random modal QBF test set. After that we describe our experiments, and the preliminary conclusions at which we have arrived.

2 The Random Modal QBF Test Set

The random modal QBF test set is based on the idea of randomly generating QBFs and then translating these into modal logic. Let's explain these two steps in more detail.

Generating QBFs. Recall that QBFs have the following shape: $\mathsf{Q}_1 v_1 \ldots \mathsf{Q}_n v_n \; CNF(v_1, \ldots, v_n)$. That is, QBFs are prenex formulas built up from proposition letters, using the booleans, and $\forall v\, \beta$ and $\exists v\, \beta$ (where v is any proposition letter).

What's involved in evaluating a QBF? We start by peeling off the outermost quantifier; if it's $\exists v$, we choose one of the truth values 1 or 0 and substitute for the newly freed occurrence of v; if it's $\forall v$, substitute both 1 and 0 for the newly freed occurrences of v. In short, while evaluating QBFs we are generating a tree, where existential quantifiers increase the depth, and universal quantifiers force branching.

In the *random modal QBF test set*, 4 parameters play a role: c, d, v, k. The parameter c is the number of clauses of the randomly generated QBF. The parameter d is the alternation depth of the randomly generated QBF; it is *not* the modal depth of the modal translation. The parameter v is the number of variables used per alternation. And k is the number of different variables used per clause.

Each clause in the matrix $CNF(v_0, \ldots, v_n)$ has k different variables and each is negated with probability *prneg* (default 0.5). The first and the third variable (if it exists) are existentially quantified. The second and fourth variable are universally quantified, etc. This aims at eliminating trivially unsatisfiable formulas. Other literals are either universal or existentially quantified variables with probability *prmod* (default 0.5). The depth of each literal is randomly chosen from 1 to d. By increasing the parameter d from odd to even, a layer of existential quantifiers is added at the beginning of the formula, and, conversely, when d increases from even to odd, a layer of universal quantifiers is added.

Translating into Modal Logic. The QBF that is produced by the random generator is translated into the basic modal logic with the usual boolean operators and $\Box$, $\Diamond$, using a variant of an encoding due to Ladner [10]. The core idea is to capture, by means of a modal formula, the 'peel off quantifiers and substitute' evaluation process for a given input QBF. The translation forces branching in the structure of the possible model whenever a universal quantifier is found in the original formula, keeps the branches separate, and makes sure there are enough modal levels in the model. It forces the structure of the possible model to be a tree, and the resulting formula is satisfiable iff the original formula is.

The parameters c, k, v and d that are used in the generation process are related to the final modal formula in the following way. The (maximum) number of clauses is $c \cdot k + (v \cdot (d+1))^2 + \lfloor v \cdot (d+1)/2 \rfloor$. The (maximum) number of proposition letters is $v \cdot (d+1)$. And the (maximum) modal depth is $v \cdot (d+1)$. These maximums obtain when $c \cdot k$ is high enough compared to $v \cdot (d+1)$ to cover all the possible proposition letters. The file size for the translated formula is linear in c, and polynomial in v and d, but usually we are not interested in big values of the last two.

3 Test Results

Settings. To evaluate the QBF test set, we used 3 satisfiability solvers for modal logic. First, we used the general first-order prover SPASS [16], version 1.0.3, extended with the layered translation of modal formulas into first-order formulas as presented in [1]. Second, we used MSPASS version V 1.0.0t.1.2.a [12]. And, third, we used *SAT version 1.3 [17]. Our experiments were run on a Pentium III 800 MHz with 128 MB of memory, running RedHat Linux 7.0. We used the November 1999 version of the QBF problem generator. To facilitate future comparisons, we used as many default settings as possible: no modal encoding, no lean encoding, the number of literals set to 4, and *prmod* and *prneg* both set to 0.5. We generated 64 instances of each problem, and the outputs of the generator were translated to the formats of the provers being used; in one case we had to convert modal formulas to first-order logic formulas. The resulting file sizes were linear in c, even though the linear coefficient varied from one solver to another.

Our main measurements concerned both CPU time elapsed (with a 10800 second timeout) and a time independent measure: the number of clauses generated for SPASS plus layering and for MSPASS, and the number of unit propagations for *SAT.[1]

Findings. We first ran the standardized tests provided by the TANCS competition: 64 instances randomly generated with $c = 20$, $v = 2$, $d = 2$, and default settings for the remaining parameters. See Figure 1.

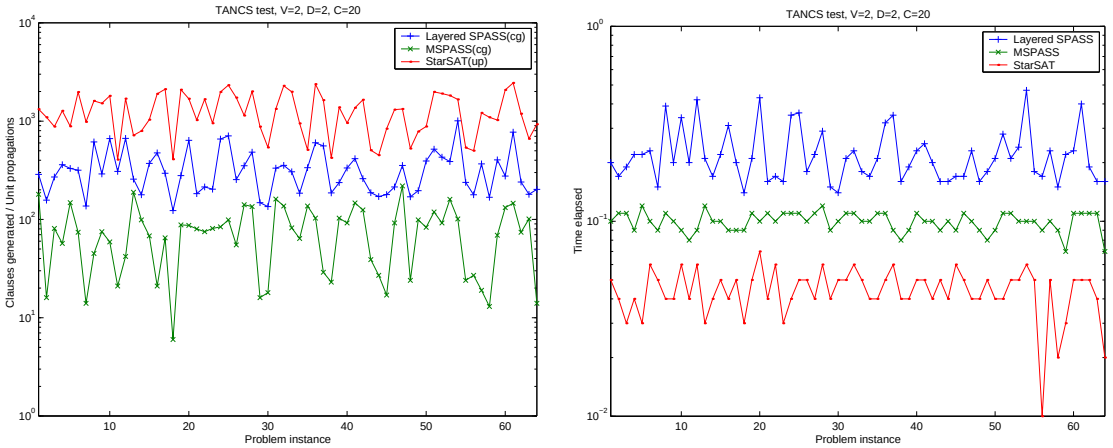

Figure 1: The standardized tests provided by TANCS, used for SPASS, MSPASS, and *SAT. (Left): clauses generated/unit propagations per problem instance, log scale. (Right): CPU time (seconds) per problem instance, log scale.

While the number of clauses generated by resolution provers and the number of unit propagations in *SAT are not directly comparable as a performance measurement, they do give an indication of the relative difficulty of a problem (or problem set). As such, we can see that the difficulty of a problem varies with the method used to solve it. The correlation between time elapsed and clauses generated/unit propagations varies widely between the methods. For this test the *SAT times are completely dominated by startup costs and don't really inform us about problem difficulty.

We ran a large number of sweeps, with each of the three provers, with $v = 2$ and increasing d from 1 to 4 (and to 5 in the case of *SAT), while increasing c from 1 to 100. The resulting CPU times and the number of clauses generated/unit propagations are depicted in Figure 2; the curves for $d = 1$, $d = 2$ do not extend to the right-hand side of the plots, as the formulas being generated with these settings are simply too small to be be able to accommodate larger number of clauses.

Many things are worth noting about Figure 2. First, the sets display an easy-hard-easy pattern familiar from propositional satisfiability testing [3]. The shape of the curves is strongly dependent on the solver

[1] We found that the number of unit propagations best describes the resource usage for each problem, as evidenced by its strong correlation to elapsed time. A detailed evaluation of unit propagations versus other statistics reported by *SAT (such as assignments found) as a suitable time independent measure, is beyond the scope of this paper.

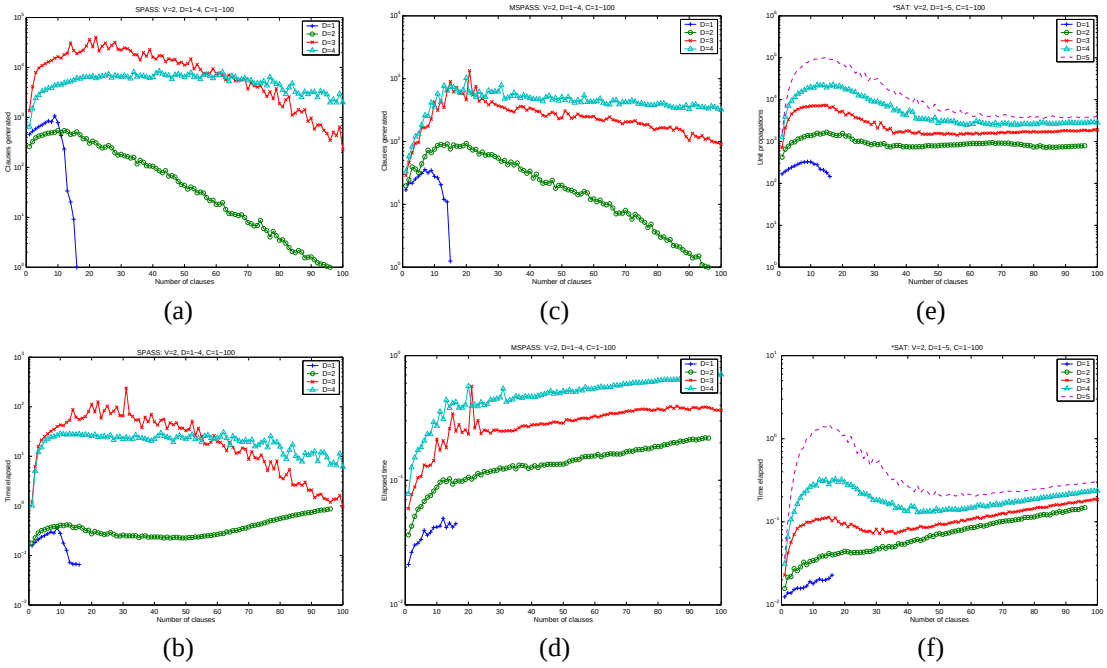

Figure 2: SPASS, MSPASS, and *SAT on QBF test sets, $v = 2$, $d = 1$–4 (5), 64 samples/point, mean values. (Top): clauses generated/unit propagations, log scale. (Bottom): CPU time in seconds, log scale.

used. Moreover, the pattern seem to vary from not-too-hard-hard-easy in some cases (SPASS, $d = 1$; MSPASS $d = 1$, $d = 2$) to not-too-hard-hard-hard in others (SPASS, $d = 4$; MSPASS, $d = 3$, $d = 4$) to not-too-hard-hard-not-too-hard in yet others (SPASS, $d = 3$; *SAT, $d = 1$–5).

Second, for SPASS and MSPASS we see that curves cross each other; this is most clearly visible in (a), where the number of clauses generated by SPASS are displayed, but it also shows up in the CPU times for SPASS (b). Hence, for SPASS (and to a lesser extent for MSPASS) the d parameter does not influence the difficulty of the problems being generated in a monotonic way. Further experimental work has shown that this 'staircase phenomenon' is also present with larger values of v for SPASS. The phenomenon is related to the special way in which QBFs grow: existential quantifiers are added to the original QBF when d is increased from odd to even, universal quantifiers when d is increased from even to odd. The former simplify matters for SPASS, while the latter make matters considerably harder.

Third, the time elapsed (displayed in (b), (d), and (f)) has a very strong dependence on file size: after the hard region has been crossed and the elapsed time tends to decrease, it actually starts going up again. The impact of input file size and I/O is most noticeable for MSPASS (plot (d)); but even in the case of *SAT, where the number of unit propagations remains more or less constant after the hard region has been traversed, the CPU times start going up: this increase is entirely due to input file size and I/O.

When increasing the parameter v, we get similar curve shapes as for $v = 2$. In Figure 3 (Left) and (Middle) we have displayed the results of running *SAT with $v = 3$. Notice that the humps indicating the hard regions are higher for $v = 3$ than for $v = 2$ (see Figure 2 (e) and (f)), indicating that the problems are harder; hence, the CPU times are not as strongly dominated by file size and I/O aspects as in the case where $v = 2$. The fact that the hard regions are 'wider' than for $v = 2$ indicates that we are not only getting harder problems, but also that the fraction of hard problems is increasing.

To which extent can we choose the difficulty of the problem and of exploring the input space? The v parameter controls the difficulty monotonically, the d parameter also with some caveats. It seems, however, that v and d do not control truly *independent* dimensions of the problem space. Combinations of v and d for which the value of $v \cdot (d + 1)$ coincides have very similar curves, as can be seen in Figure 3 (Right). This suggests that $v \cdot (d + 1)$ is the correct dimension along which the QBF problem space should be explored. This comes as no surprise: as we saw in Section 2, $v \cdot (d + 1)$ determines both the maximum

number of propositional variables and the maximum modal depth in the translated formula. (As an aside, with increasing values of $v \cdot (d+1)$, the truly hard region for a given setting of parameters moves to the right as we increase the number of clauses.)

An important aspect that we have not discussed so far is the satisfiable vs. non-satisfiable fraction. The parameter c does indeed allow us to control the satisfiability fraction: it goes from 1 to 0 monotonically with c. However, there are remarkably few values of c for which the satisfiable fraction is 1; see Figure 4. As illustrated by Figure 4 (top left), we have found satisfiable fractions of about 20% in many repeated runs of the standardized 20/2/2 TANCS test (see Figure 1). Moreover, there is a heavy 'tail' of unsatisfiable problems, as indicated by the curves in Figure 3 (Right). Contrary to intuition, the constrainedness of problems does not seem to depend very strongly on the d parameter; for a fixed v, increasing d from even to odd doesn't shift the satisfiable fraction graph by any noticeable amount. The constrainedness of the underlying models, then, remains unchanged despite the addition of variables and the increase in depth.

There's one more thing worth pointing out. The 50% satisfiable mark occurs towards the right-hand side of the truly hard region; this may suggest that the hardest problems within a given test set are likely to be satisfiable ones.

Finally, recall that a modal formula is *trivially satisfiable* iff it is satisfiable on a model with a single node [8, 9]. Clearly, trivial satisfiability is not a problem for modal QBF test sets. For a start, very few satisfiable problems are generated anyway, and because of the highly structured form of the randomly generated QBFs, the resulting modal formulas always contain $\Diamond$-subformulas, thus avoiding trivial satisfiability.

4 What Have We Learned So Far?

For an assessment of the random modal QBF test set by means of the general criteria for evaluating modal test methodologies that were put forward by Horrocks, Patel-Schneider, and Sebastiani [8], we refer the reader to [6]; see also [14]. Here we adopt a different perspective: what part of the problem space do we cover with the random modal QBF generator?

The modal QBF test set seems to represent just a restricted subarea of the whole input space. There seem to be three reasons for this. First, the QBF test set provides poor coverage of the satisfiable region, and especially of the easily satisfiable region; most of the modally encoded QBF-formulas generated with values of v and d that are within reach of today's tools, are hard and unsatisfiable, as suggested by Figure 4. Second, the modally encoded QBFs are of a very special shape, which seems to lead to the so-called staircase phenomenon for some solvers. And third, the v and d parameters end up being substantially overlapping and interrelated as part of the translation of QBFs into modal formulas. A strong point in favor of the QBF test set is that it is possible to generate hard problems with a large modal depth which are still within reach of today's modal satisfiability solvers; in this respect the QBF random test methodology fares better than the New_3CNF$_{\Box_m}$ test methodology, as reported in [8].

In sweeps with increasing values of c, the phase transition already occurs for small values of c, so that explorations of larger values of c reveals no novelties, and the effective search space is relatively small. More generally, while QBFs are good representatives of the class of all quantified boolean formulas, the

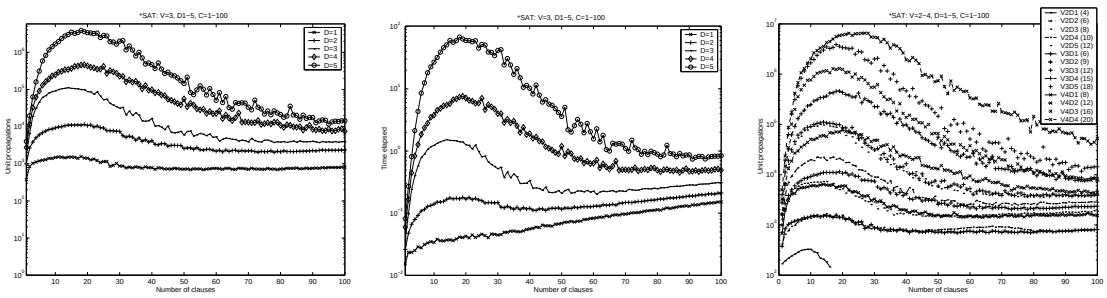

Figure 3: *SAT results, 64 samples/point, mean values, log scale. (Left): $v = 3$, $d = 1$–5, unit propagations. (Middle): $v = 3$, $d = 1$–5, CPU time in seconds. (Right): $v = 2$–4, $d = 1$–5, and $c = 1$–100; the numbers in brackets indicate the value of $v \cdot (d+1)$.

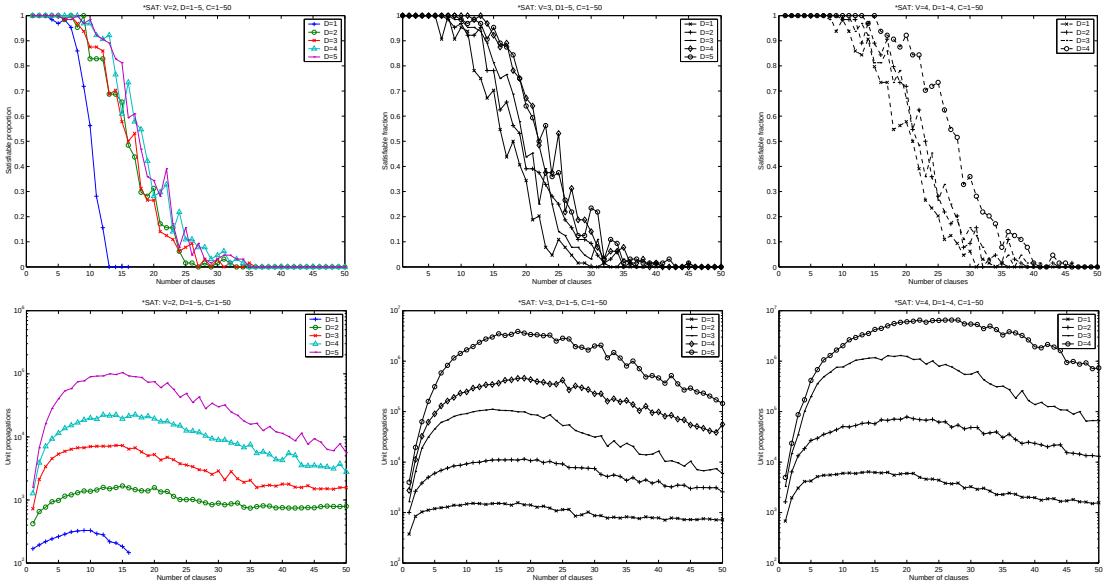

Figure 4: *SAT results for $d = 1$–4 (5), $c = 1$–50, 64 samples/point, mean values. (Top): Satisfiable fraction. (Bottom): Unit propagations. (Left): $v = 2$. (Middle): $v = 3$. (Right): $v = 4$.

class of modally encoded QBFs are restricted to formulas whose underlying models have a very regular structure (see [6] for elaborations on this point). As a consequence, the comment (made in the introduction) about the naturalness of the modal QBF test set may only be partially true.

5 Next Steps

Our ongoing work is organized in a number of stages. At present we're collecting data for the provers that we haven't considered so far (DLP [13] and RACER [5]). We're particularly keen on finding out to which extent the shape of our plots is dependent on differences in provers. This, we hope, will give us more insight into whether our results are really about the modal satisfiability problem, or the particular algorithms and implementations, or just about the particular kind of formulas being generated. Another way in which we want to increase our understanding of the nature of the modal QBF test set is by developing suitable ways of comparing the random QBF generator and the (2001 version of the) $\text{CNF}_{\Box_m}$ generator.

As an aside, we want to see whether we can restrict ourselves to investigations along the $v \cdot (d + 1)$ dimension only (instead of each of v and d) — not only to save on (months worth of) CPU time, but also to get a better understanding of the part of the problem space that we are actually exploring.

Finally, we are interested in compare our modal results against results obtained by feeding the (untranslated) QBFs to QBF solvers, so as to get a better understanding of the effect of the encodings of QBFs into modal formulas.

References

[1] C. Areces, R. Gennari, J. Heguiabehere, and M. de Rijke. Tree-based heuristics in modal theorem proving. In W. Horn, editor, *Proc. ECAI 2000*, 2000.

[2] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. In *Proc. KR-92*, 1992.

[3] I. Gent, H. van Maaren, and T. Walsh, editors. *SAT 2000*. IOS Press, 2000.

[4] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures—the case study of modal K. In *Proc. CADE-96*, 1996.

[5] V. Haarslev and R. Möller. RACER system description. In *Proc. IJCAR 2001*, 2001.

[6] J. Heguiabehere and M. de Rijke. The random modal QBF test set. In *Proc. IJCAR Workshop on Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, 2001.

[7] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, and S4. Technical report IAM-96-015, University of Bern, Switzerland, 1996.

[8] I. Horrocks, P.F. Patel-Schneider, and R. Sebastiani. An analysis of empirical testing for modal decision procedures. *Logic Journal of the IGPL*, 8:293–323, 2000.

[9] U. Hustadt and R. A. Schmidt. On evaluating decision procedures for modal logic. In *Proc. IJCAI-97*, pages 202–207, 1997.

[10] R. Ladner. The computational complexity of provability in systems of modal logic. *SIAM Journal on Computing*, 6:467–480, 1977.

[11] F. Massacci. Design and results of the Tableaux-99 non-classical (modal) system competition. In *Proc. Tableaux'99*, 1999.

[12] MSPASS V 1.0.0t.1.2.a. URL: `http://www.cs.man.ac.uk/~schmidt/mspass`. Accessed February 23, 2001.

[13] P.-F. Patel-Schneider. DLP system description. In *Proc. DL'98*, 1998.

[14] P.-F. Patel-Schneider and R. Sebastiani. A new very-general method to generate random modal formulae for testing decision procedures. Submitted, 2001.

[15] J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[16] SPASS Version 1.0.3. URL: `http://spass.mpi-sb.mpg.de/`. Accessed May 23, 2000.

[17] A. Tacchella. *SAT system description. In *Proc. DL'99*, 1999.

[18] TANCS: Tableaux Non-Classical Systems Comparison. `http://www.dis.uniroma1.it/~tancs`. Accessed January 17, 2000.