Evaluating Automated Theorem Provers for First-Order Modal Logics

Thomas Raths* Jens Otten

Institut für Informatik, University of Potsdam
August-Bebel-Str. 89, 14482 Potsdam-Babelsberg, Germany
{traths,jeotten}@cs.uni-potsdam.de

Abstract

First-order modal logics have many applications, e.g., in planning, natural language processing, program verification, querying knowledge bases, and modeling communication. This paper gives an overview of several new implementations of theorem provers for first-order modal logics based on different proof calculi. Among these calculi are the standard sequent calculus, a prefixed tableau calculus, an embedding into simple type theory, and an instance-based method. All these theorem provers are tested and evaluated on the QMLTP problem library for first-order modal logic. The results of these test runs are compared and analyzed.

1 Introduction

Modal logics extend classical logic with the modalities ”it is necessarily true that” and ”it is possibly true that” represented by the unary operators $\Box$ and $\Diamond$, respectively. *First-order* modal logics extend propositional modal logics by *domains* specifying sets of objects that are associated with each world, and the standard universal and existential quantifiers [9, 15, 17].

First-order modal logics allow a natural and compact knowledge representation. The subtle combination of the modal operators and first-order logic enables specifications of epistemic, dynamic and temporal aspects, and of infinite sets of objects. For this reason, first-order modal logics have many applications, e.g., in planning, natural language processing, program verification, querying knowledge bases, and modeling communication. In these applications modalities are used to represent incomplete knowledge, programs, or to contrast different sources of information. First-order components, such as variables, functions, predicates and quantifiers enable to describe objects, their properties, types, and abstract information that can be instantiated later.

For example, the planning system PKS [27] constructs conditional plans. It uses modal operators to represent incomplete knowledge, constants and predicates to describe objects and their properties, and variables and functions to generate abstract plans, which are instantiated later, when sufficient knowledge is available. An inference procedure for a restricted quantified modal logic determines whether the plan achieves

*This work is partly funded by the German Science Foundation DFG under reference number KR858/9-1.

the goal and the preconditions of the actions hold, and generates the effects of the actions. PKS can be applied to, e.g., dialogue planning [37]. The dialogue system Artimis [33] and the sentence-planner SPUD [40] plan, generate and interpret sentences in a natural language. They use modalities to distinguish beliefs, intentions and actions of the system and the user. First-order logic components represent objects, properties and quantified statements. Variables enable to process abstract instructions that can be instantiated later when more information is available [39]. An inference engine is adapted to plan and interpret the sentences. The systems KIV [31], VSE-II [1] and KeY [8] are advanced tools for program verification and synthesis. Their proof components use dynamic and temporal first-order logic which are closely related to first-order modal logic. The modalities represent programs, whereas functions, variables and quantifiers characterize attributes, types and the creation of objects. Likewise the verification of database update programs [36] and the integration of UML specification [10] can be described in first-order modal logic. A first-order modal logic is also used as query language for description logic knowledge bases [11]. Automated reasoning is required to answer queries and to verify and optimize integrity conditions. Finally, first-order modal logics are used to describe communication and cooperation [12, 23].

All these applications require the use of automated theorem proving (ATP) systems for *first-order* modal logics. Whereas there are some ATP systems available for propositional modal logics, e.g., MSPASS [20] and modleanTAP [4], there are currently only few ATP systems that can deal with the full first-order fragment of modal logics.

The purpose of this paper is to introduce some new ATP systems for first-order modal logics and to evaluate, compare and analyze their performance on a standardized problem set. The reader is assumed to be familiar with the syntax and semantics of first-order modal logics, see, e.g., [15, 17]. If not stated otherwise the standard semantics and the following options regarding first-order terms for all evaluated ATP systems are considered: term designation is rigid, i.e., the terms denote the same object in each world, and terms are assumed to be local, i.e., any ground term denotes an existing object for each world.

This paper is structured as follows. In Section 2 all new and existing ATP systems for first-order modal logics are shortly described. Section 3 provides details about the used problem set and presents comprehensive performance results and comparisons of all described ATP systems. Section 4 concludes with a short summary and a few remarks on future work.

2 Implementing First-Order Modal Theorem Provers

The following (new and existing) ATP systems for first-order modal logics are described in this section: MleanSeP based on the standard sequent calculus, GQML-Prover and MleanTAP based on tableau calculi, M-Leo-II and M-Satallax based on an embedding into simple type theory and the f2p-MSPASS based on an instance-based method. Table 1 gives an overview of these systems.

2.1 Sequent Calculus

MleanSeP implements the standard sequent calculus for several modal logics.[1] It is implemented in Prolog. Proof search is carried out in an analytic way and free-variables are used in combination with a dynamic Skolemization that is calculated during the

[1]MleanSeP can be downloaded at `www.leancop.de/mleansep/programs/mleansep11.pl` .

actual proof search. Together with the occurs-check of the term unification algorithm this ensures that the Eigenvariable condition is respected.

MleanSeP 1.1 can deal with the first-order cumulative and constant domains of the modal logics K, K4, D, D4, S4, and T. To deal with constant domains, the *Barcan formula*[2] is automatically added to the given formula in a preprocessing step.

2.2 Tableau Calculi

GQML-Prover [44] is based on a free-variable tableau calculus using annotated tableau nodes and function symbols. It uses a liberalized δ^{+}-rule and is implemented in OCaml. GQML-Prover 1.2 can deal with cumulative, constant and varying domains of the modal logics K, K4, D, S4, and T, using rigid or non-rigid terms, and local or non-local terms.

MleanTAP implements a prefixed tableau calculus.[3] The compact system is implemented in Prolog. It uses not only free term variables but also free string variables for the prefixes and a prefix unification procedure. It is based on the ileanTAP system for first-order intuitionistic logic [24]. At first MleanTAP performs a purely classical proof search. After a classical proof is found, the prefixes of those literals that close branches in the (classical) tableau are unified. The existence of a prefix substitution ensures that the given formula is valid in modal logic as well [45]. If no prefix substitution exists backtracking is done in order to find alternative classical proofs (and prefixes). To deal with different modal logics only the prefix unification procedure has to be adapted [22].

MleanTAP 1.1 can deal with the first-order cumulative and constant domains of the modal logics D and S4. By further modifying the prefix unification algorithm MleanTAP can be extended to the modal logics D4, S5, and T.

2.3 Embedding into Simple Type Theory

M-Leo-II 1.2 and M-Satallax 1.4 extend the ATP systems Leo-II 1.2 and Satallax 1.4 to first-order modal logics, respectively. Both provers use an embedding of quantified modal logic into simple type theory [6]. Leo-II [7] and Satallax [2] are ATP systems for typed higher-order logic.[4] Leo-II is based on an extensional higher-order resolution calculus. It cooperates with a first-order ATP system, by default E [34], and applies term sharing and term indexing techniques. It is implemented in OCaml. Satallax uses a complete ground tableau calculus for higher-order logic to generate successively propositional clauses and calls MiniSat repeatedly to test unsatisfiability of these clauses. It can be regarded as an instance-based method for higher-order logic. Satallax is implemented in Steel Bank Common (SBC) Lisp.

Currently the embedding of quantified modal logic into simple type theory works for constant domains only. Thus, M-Leo-II 1.2 and M-Satallax 1.4 can deal with the first-order constant domains of the modal logics K, D, S4, S5, and T.

[2]The Barcan formula scheme has the form $\forall\vec{x}(\Box p(\vec{x}) \Rightarrow \Box\forall\vec{x}p(\vec{x})$ with $\vec{x} = x_1, \ldots, x_n$ for all predicates p with $n \geq 1$.

[3]MleanTAP can be downloaded at `www.leancop.de/mleantap/programs/mleantap11.pl` .

[4]These two higher-order ATP systems were selected as they have the best performance of all currently available systems for higher-order logic [43].

2.4 Instance-Based Method

f2p-MSPASS 3.0 uses an instance-based method to generate ground formulas and the ATP system MSPASS 3.0 for proving formulas in propositional modal logic. Like most instance-based methods, the f2p-MSPASS system consists of two components. The first component, called first2p, takes a first-order modal formula, removes all quantifiers and replaces every variable with a unique constant. The second component, MSPASS [20], takes the resulting (ground) formula and tries to find a proof or a counter model. If a counter model is found first2p adds quantified subformulas to the original formula and instantiates variables with new terms. Afterwards MSPASS is again used to find a proof for the resulting formula. If first2p is unable to add any new instances of subformulas, the original formula is invalid.

first2p is written in Prolog. It does not translate the given formula into any clausal form but preserves its structure throughout the whole proof process. Due to the restrictions of modal logics this instance-based approach does only work for formulas that contain either only existential or only universal quantifiers. MSPASS is an extension of and incorporated into the resolution-based ATP system SPASS. It uses several translation methods into classical logic. By default the standard relational translation from modal logic into classical logic is applied.

f2p-MSPASS 3.0 and can deal with the first-order cumulative and constant domains of the modal logics K, K4, K5, B, D, S4, S5, and T. To deal with constant domains, first2p automatically adds the Barcan formula (see Section 2.1) to the original formula in a preprocessing step.

Table 1: First-order modal ATP systems

ATP system	modal logics	domains	equality	language
MleanSeP 1.1	K,K4,D,S4,T	cumulative, constant	no	Prolog
MleanTAP 1.1	D, S4	cumulative, constant	no	Prolog
GQML-Prover 1.2	K,K4,D,S4,T	cumul., const., vary.	no	OCaml
M-Leo-II 1.2	K,D,S4,S5,T	constant	yes	SBC Lisp
M-Satallax 1.4	K,D,S4,S5,T	constant	yes	OCaml
f2p-MSPASS 3.0	K,K4,K5,B, D,S4,S5,T	cumulative, constant	no	C/Prolog

3 Evaluating First-Order Modal Theorem Provers

In this section the ATP systems presented in Section 2 are evaluated on the QMLTP library for first-order modal logics. Some details about the QMLTP library are given first before comprehensive performance results and comparisons are presented.

3.1 Problem Set: The QMLTP Library

Testing ATP systems using standardized problem sets is a well-established method for measuring their performance. For example, the TPTP library [41] and the ILTP library [30] were developed for classical and intuitionistic logic, respectively. These problem libraries have fostered the development of more efficient systems for these logics. The basic idea of using problem libraries is to run the ATP systems on the problems included in the library, determine the number of problems solved within a given time limit, and to collect further data, e.g., the average run time used to solve the problems. In order to have these performance results reflect the capabilities of the systems accurately, the problem library shall be large enough, span a variety of difficulty and subject matters, and have a standardized syntax [41].

Until recently there existed only very small collections of formulas that could be used for testing and evaluating ATP systems for *first-order* modal logics. A small set of first-order formulas was used for testing GQML-Prover [44]. For *propositional* modal logics there exist some scalable problem classes [3] and approaches that generate formulas randomly in a normal form [26]. But generating formulas randomly is not an appropriate approach for the first-order case.

The Quantified Modal Logic Theorem Proving (QMLTP) library [29] provides a comprehensive set of standardized problems in first-order modal logics and, thus, constitutes a convenient basis for testing and evaluating the performance of ATP systems for first-order modal logics. The main purpose of the QMLTP library is to stimulate the development of new ATP systems and calculi for first-order modal logics. The current release v1.0 of the QMLTP library contains 500 problems represented in an extended TPTP syntax. The problem set is available for download at `http://www.iltp.de/qmltp`.

Release v1.0 of the QMLTP library includes 245 problems that are generated by using Gödel's embedding of intuitionistic logic into the modal logic S4 [19]. The original problems were taken from the TPTP library [41]. 10 problems were taken from applications, e.g., planning, querying databases, natural language processing and communication, and software verification [10, 11, 13, 32, 38, 39]. 175 problems come from various textbooks [14, 15, 16, 18, 28, 35, 44] and 70 problems from the TANCS-2000 system competition for modal ATP systems [21].

There are only few problems from real applications in the current release of the QMLTP library. Future versions will include more problems from applications mentioned in Section 1 once they are submitted to the QMLTP library. The aim of the QMLTP library is to start a cycle in which developers are inspired to improve their ATP systems and users are encouraged to apply these systems and to contribute their problems to the library stimulating the development of more efficient systems.

Each problem in the QMLTP library is assigned a modal status and a modal rating. The *status* is either `Theorem`, `Non-Theorem` or `Unsolved`. Problems with `Unsolved` status have not been solved by any ATP system.[5] The *rating* determines the difficulty of a problem with respect to current state-of-the-art ATP systems. It is the fraction of state-of-the-art ATP systems that are *not* able to solve a problem within a given time limit. For example a rating of 0.3 indicates that 30% of the state-of-the-art systems do *not* solve the problem; a problem with rating of 1.0 cannot be solved by any state-of-the-art system. A *state-of-the-art* system is an ATP system whose set of solved problems is not subsumed by that of any another ATP system. In the current release of the QMLTP library status and rating information is given with respect to the constant and cumulative domains of the modal logics D and S4.

In order to represent modal problems in a standardized *syntax*, the Prolog syntax of the TPTP library [41] is extended by the modal operators $\Box$ and $\Diamond$. The two Prolog atoms "`#box`" and "`#dia`" are used for representing $\Box$ and $\Diamond$, respectively. The formulas $\Box F$ and $\Diamond F$ are then represented by "`#box:F`" and "`#dia:F`", respectively

[5]No theoretical investigations regarding the status of formulas have been done.

```
%------------------------------------------------------------------------
% File     : SYM001+1 : QMLTP v1.0
% Domain   : Syntactic (modal)
% Problem  : Barcan scheme instance. (Ted Sider's qml wwf 1)
% Version  : Especial.
% English  : if for all x necessarily f(x), then it is necessary that for
%            all x f(x)
% Refs     : [Sid09] T. Sider. Logic for Philosophy. Oxford, 2009.
%          : [Brc46] [1] R. C. Barcan. A functional calculus of first
%            order based on strict implication. Journal of Symbolic Logic
%            11:1-16, 1946.
% Source   : [Sid09]
% Names    : instance of the Barcan formula
%
% Status   :       cumulative  constant
%            D    Unsolved    Theorem       v1.0
%            S4   Unsolved    Theorem       v1.0
%
% Rating   :       cumulative  constant
%            D    1.00         0.00        v1.0
%            S4   1.00         0.00        v1.0
%
% term conditions for all terms:  designation: rigid,  extension: local
%
% Comments :
%------------------------------------------------------------------------
qmf(con,conjecture,
(( ! [X] : (#box : ( f(X) ) ) ) => (#box : ( ! [X] : ( f(X) ) )))).
%------------------------------------------------------------------------
```

Figure 1: Example of problem file `SYM001+1` of the QMLTP library.

(see also Figure 1). For future extensions to multi-modal logic these atoms can be extended to, e.g., Prolog terms of the form "`#box(i)`" or "`#dia(i)`" in which the index "`i`" is an arbitrary Prolog atom. As there exists no ATP system for first-order multi-modal logic, the current release of the QMLTP library is restricted to uni-modal problems only.

A header with useful information is added to the *presentation* of each problem. It is adapted from the TPTP library and includes information about the file name, the problem description, the modal status and the modal difficulty rating. An example file of a first-order modal problem is given in Figure 1.

Note that the problem files of the QMLTP library are primary intended to present the *syntax* of modal formulas. The options of the intended *semantics*, e.g., the interpretation of the modal operators in the different modal logics is left to the (user of the) particular ATP system.[6] This increases the flexibility of the library as it can be used for all (uni-modal) logics that share the standard modal syntax.

3.2 Performance Evaluation

Several criteria are used to evaluate and compare the performance of ATP systems. The main measure is the number of problems that an ATP system solves within a given time limit. For the evaluation the output of a proof or a counter model is not required, i.e., an assurance of the existence of a proof or counter model is sufficient. The time limit is in terms of CPU time since no extensive memory usage was observed and, thus, wall clock time is not significantly higher than CPU time. The average CPU runtime of an ATP system was determined only for problems solved by *all* systems in order to make the comparison fair. Otherwise, a system that spends more time solving a difficult problem that is not solved by other systems would be disadvantaged.

[6]Even though some problems, e.g., problems stemming from Gödel's embedding were originally developed with a specific modal logic in mind.

Another interesting property when evaluating ATP systems is their time complexity behavior, i.e., the increase of number of solved problems when increasing the time limit. When an ATP system solves most problems of the ones it can solve at all within a second, its time complexity behavior is worse than that of a systems that still solves a significant number of problems after 10 or 100 seconds. A very steep time complexity behavior indicates that the ATP system's underlying proof calculus needs to be improved. What kind of problems are solved is interesting as well. Problems with equality are of interest, because some calculi and techniques might be more appropriate to deal with equality than others.

There are three further criteria to rate the performance of ATP systems: the state-of-the-art (SOTA) system rating, the state-of-the-art contribution (SOTAC) and the efficiency measure [41, 42]. The *SOTA system rating* states how many difficult problems an ATP system can solve. It is the fraction of problems that can be solved by the regarded system but not by all systems. The value is 1.0 if the system can solve all difficult problems, and is 0.0 if it can *only* solve problems that are solved by all (state-of-the-art) systems as well. The *SOTAC* of an ATP system measures its unique problems solving capability. The SOTAC of a specific *problem* is the inverse of the number of state-of-the-art systems that solve the problem. For example, the maximal value 1.0 indicates that only one system solves the problem, 0.5 means that two state-of-the-art system solve the problem. The SOTAC of an ATP *system* is the average SOTAC over all problems solved by the system. The less "unique" an ATP system the smaller is its SOTAC. Finally, the *efficiency measure* takes the number of solved problems and time taken to solve them into account. It is the fraction of solved problems divided by the average time needed to solve these problems. The more problems are solved and the faster they are solved by an ATP system the higher is its efficiency measure.

3.3 The Test Environment

The ATP systems described in Section 2 are evaluated on the modal logics D and S4 with constant and cumulative domains. These are the modal logics supported by the majority of available ATP systems. The standard semantics for the modal logics D and S4, rigid term designation and local terms are used [15].

Table 2: Test Environment

hardware	3.4 GHz Xeon, 4 GB RAM	
operating system	Linux 2.6.24-24.x86_64	
time limit	600 sec.	
modal logic	D	S4
accessibility relation	serial	reflexive, transitive
axioms	$\Box A \rightarrow \Diamond A$	$\Box A \rightarrow A$, $\Box A \rightarrow \Box\Box A$
domains	cumulative, constant	
terms	designation: rigid, extension: local	

A test environment was developed for automatically conducting all performance tests and for collecting and evaluating the results of all test runs. These test runs were conducted on an eight-processor cluster system in order to simultaneously test several ATP systems at a time. All ATP systems and components written in Prolog use

ECLiPSe Prolog 5.10. For M-Satallax 1.4 and M-Leo-II 1.2 the binaries of the CASC-J5 [43] were used. For MSPASS the sources of SPASS 3.0 were compiled using the GNU gcc compiler version 4.2.4.

The CPU time limit for all proof attempts was set to 600 seconds. For handling equality the equality axioms were added in a preprocessing step for the ATP systems MleanSeP, MleanTAP, GQML-Prover and f2p-MSPASS. The time required for adding the equality axioms is less than a second and not included in the overall timings. Table 2 summarizes the test conditions.

3.4 Performance Statistics

Table 3 and Table 4 give an overview of the test results for all ATP systems described in Section 2. M-Satallax and M-Leo-II can be applied to the constant domains only.

Table 3: Number of proved problems of the QMLTP library v1.0

	D		S4	
	cumulative	constant	cumulative	constant
MleanSeP 1.1	117	120	203	201
MleanTAP 1.1	84	120	189	205
M-Satallax 1.4	-	107	-	188
M-Leo-II 1.2	-	104	-	172
GQML-Prover 1.2	88	95	137	133
f2p-MSPASS 3.0	47	47	88	88

Table 4: Number of found counter models of the QMLTP library v1.0

	D		S4	
	cumulative	constant	cumulative	constant
MleanSeP 1.1	1	1	1	1
MleanTAP 1.1	1	1	1	1
M-Satallax 1.4	-	7	-	71
M-Leo-II 1.2	-	0	-	0
GQML-Prover 1.2	0	0	0	0
f2p-MSPASS 3.0	108	107	42	36

Table 5, 7, 9, and 11 present the performance results for the modal logics D and S4 with cumulative and constant domains, respectively. They contain the number of solved problems, the number of proved problems, and the number of counter models (disproved) found within the time limit, the fraction of solved problems, the number of solved problems within a specific time interval, the number of time outs, the number of solved problems containing equality, the number of problems solved by only one ATP system, the SOTA system rating, the SOTAC, and the efficiency measure as described in section 3.2. The average run time was determined only for problems solved by *all* ATP systems. These are 9% and 16% of all problems for the modal logics D and S4, respectively.

For some problems MleanSeP and MleanTAP produce a stack overflow (stack). f2p-MSPASS cannot be applied to 299 problems (gave up) as these problems contain both existential and universal quantifiers (see remarks in Section 2.4). For some problems GQML-Prover returns wrong results (inconsistent).[7]

Table 6, 8, 10, and 12 show the number of problems solved by one ATP system but not by another system. For example, for the modal logic D with cumulative domains MleanSeP solves 34 problems that are not solved by MleanTAP (see Table 6). The performance graph of all considered ATP systems for the modal logics D and S4 with cumulative and constant domains are depicted in Figure 2, 3, 4, and 5, respectively.

To compare the performance of the modal ATP systems with an ATP system for classical logic, the classical prover leanTAP [5] was run on all modal problems, in which the modal operators have been removed. Out of the 500 problems leanTAP 2.3 proves 282 problems and finds a counter model for one problem. It solves all except one problem within one second.

Table 5: Benchmark results for modal logic D with cumulative domains

	MleanSeP 1.1	MleanTAP 1.1	GQML-Prover 1.2	f2p-MSPASS 3.0
solved	118	85	88	155
[%]	24%	17%	18%	31%
proved	117	84	88	47
disproved	1	1	0	108
0s to 1s	117	85	71	155
1s to 10s	0	0	1	0
10s to 100s	0	0	16	0
100s to 600s	1	0	0	0
time out	349	411	143	46
stack / gave up	33	4	264	0
not applicable	0	0	0	299
inconsistent	0	0	5	0
solved with equality	37	18	16	0
only by this system	27	0	22	107
average time [s]	<0.01	<0.01	0.01	0.01
SOTA system rating	0.26	0.16	0.17	0.38
SOTAC	0.49	0.32	0.51	0.76
efficiency measure	0.05	289.00	0.02	102.23

Table 6: Number of problems solved by A but *not* by B for D with cumulative domains

system A	system B			
	MleanSeP 1.1	MleanTAP 1.1	GQML-Prover 1.2	f2p-MSPASS 3.0
MleanSeP 1.1	0	34	53	71
MleanTAP 1.1	1	0	26	37
GQML-Prover 1.2	28	34	0	55
f2p-MSPASS 3.0	108	107	117	0

[7] An inconsistency occurs, e.g., for problem SYM176+1 for all considered modal logics.

Table 7: Benchmark results for modal logic D with constant domains

	MleanSeP 1.1	MleanTAP 1.1	M-Satallax 1.4	M-Leo-II 1.2	GQML-Prover 1.2	f2p-MSPASS 3.0
solved	121	121	114	104	95	154
[%]	24%	24%	23%	21%	19%	31%
proved	120	120	107	104	95	47
disproved	1	1	7	0	0	107
0s to 1s	93	118	97	98	79	154
1s to 10s	27	3	10	0	1	0
10s to 100s	0	0	2	1	15	0
100s to 600s	1	0	5	5	0	0
time out	346	377	386	396	142	47
stack / gave up	33	2	0	0	257	0
not applicable	0	0	0	0	0	299
inconsistent	0	0	0	0	6	0
solved with equality	36	28	12	10	16	0
only by this ATP	6	4	4	1	15	104
average time [s]	<0.01	<0.01	0.34	0.05	<0.01	0.01
SOTA system rating	0.15	0.15	0.14	0.13	0.11	0.21
SOTAC	0.29	0.28	0.26	0.23	0.38	0.72
efficiency measure	0.04	2.08	0.04	0.03	0.02	38.25

Table 8: Number of problems solved by A but *not* by B for D with constant domains

system A	system B					
	MleanSeP 1.1	MleanTAP 1.1	M-Satallax 1.4	M-Leo-II 1.2	GQML-Prover 1.2	f2p-MSPASS 3.0
MleanSeP 1.1	0	14	31	35	53	74
MleanTAP 1.1	14	0	29	31	48	73
M-Satallax 1.4	24	22	0	12	45	64
M-Leo-II 1.2	18	14	2	0	37	58
GQML-Prover 1.2	33	28	32	34	0	63
f2p-MSPASS 3.0	107	106	104	108	116	0

3.5 Comparison of Performance Results

In general, ATP systems prove more problems of the QMLTP library with respect to the modal logic S4 than with respect to the modal logic D. Furthermore, more problems are proved for the constant domains condition than for the cumulative domains condition.

These results are in line with the fact that every formula that is valid in the modal logic D is also valid in S4, and that every formula that is valid for the cumulative domains condition is also valid for the constant domains condition as shown in the following figure:

Table 9: Benchmark results for modal logic S4 with cumulative domains

	MleanSeP 1.1	MleanTAP 1.1	GQML-Prover 1.2	f2p-MSPASS 3.0
solved	204	190	137	130
[%]	41%	38%	27%	26%
proved	203	189	137	88
disproved	1	1	0	42
0s to 1s	184	186	126	129
1s to 10s	8	2	0	0
10s to 100s	8	1	10	1
100s to 600s	4	1	1	0
time out	262	306	270	71
stack / gave up	34	4	91	0
not applicable	0	0	0	299
inconsistent	0	0	2	0
solved with equality	51	29	6	1
only by this system	28	17	28	48
average time [s]	0.72	<0.01	0.18	0.01
SOTA system rating	0.31	0.28	0.16	0.15
SOTAC	0.44	0.41	0.44	0.53
efficiency measure	0.06	0.23	0.06	1.28

Table 10: Number of problems solved by A but *not* by B for S4 with cumulative dom.

system A	system B			
	MleanSeP 1.1	MleanTAP 1.1	GQML-Prover 1.2	f2p-MSPASS 3.0
MleanSeP 1.1	0	32	95	123
MleanTAP 1.1	18	0	85	108
GQML-Prover 1.2	30	34	0	74
f2p-MSPASS 3.0	49	48	65	0

$$\begin{array}{ccc} \{F\mid F \text{ is valid in D cumulative domains}\} & \subset & \{F\mid F \text{ is valid in D constant domains}\} \\ \subset & & \subset \\ \{F\mid F \text{ is valid in S4 cumulative domains}\} & \subset & \{F\mid F \text{ is valid in S4 constant domains}\}. \end{array}$$

However, for the modal logic S4 MleanSeP proves more problem for the cumulative domains than for the constant domains. The reason for this behavior is that the inclusion of the Barcan formula increases the search space for formulas that are valid under both domain conditions.

MleanSeP proves the highest number of problems, except for S4 constant domain where MleanTAP proves more problems than any other prover. MleanSeP also proves the highest number of problems containing equality for all considered modal logics.

In general MleanTAP proves only slightly fewer problems than MleanSeP. It has the best performance for S4 with constant domains and proves many problems with equality as well. The time complexity behavior of MleanTAP is worse than that of MleanSeP. Both MleanSeP and MleanTAP each found only one counter model.

Table 11: Benchmark results for modal logic S4 with constant domains

	MleanSeP 1.1	MleanTAP 1.1	M-Satallax 1.4	M-Leo-II 1.2	GQML-Prover 1.2	f2p-MSPASS 3.0
solved	202	206	259	172	133	124
[%]	40%	41%	52%	34%	27%	25%
proved	201	205	188	172	133	88
disproved	1	1	71	0	0	36
0s to 1s	170	203	166	155	123	123
1s to 10s	20	2	51	7	0	0
10s to 100s	8	1	28	4	9	1
100s to 600s	4	0	14	6	1	0
time out	265	292	241	328	271	77
stack / gave up	33	2	0	0	83	0
not applicable	0	0	0	0	0	299
inconsistent	0	0	0	0	13	0
solved with equality	45	29	14	9	4	1
only by this system	14	10	35	1	8	0
average time [s]	0.05	<0.01	0.41	0.08	0.18	0.01
SOTA system rating	0.16	0.16	0.22	0.12	0.08	0.07
SOTAC	0.30	0.29	0.35	0.22	0.29	0.29
efficiency measure	0.05	2.61	0.04	0.05	0.06	1.08

M-Satallax and M-Leo-II prove only slightly fewer problems than MleanSeP and MleanTAP for D and S4 with constant domains. For the modal logic S4 M-Satallax is very strong in finding counter models and solves a large number of problems that were taken from the TANCS-2000. M-Leo-II does not find counter models and its time complexity is slightly worse than that of M-Satallax.

f2p-MSPASS finds a high number of counter models for all considered modal logics. Like M-Satallax it solves a high number of TANCS-2000 problems. However, its time complexity behavior is very steep. All except one problem are solved within one second. f2p-MSPASS cannot be applied to 299 problems (see remarks in Section 2.4). There are only 31 non-propositional modal problems that are suitable for the instance-

Table 12: Number of problems solved by A but *not* by B for S4 with constant domains

system A	system B					
	MleanSeP 1.1	MleanTAP 1.1	M-Satallax 1.4	M-Leo-II 1.2	GQML-Prover 1.2	f2p-MSPASS 3.0
MleanSeP 1.1	0	23	43	54	90	122
MleanTAP 1.1	27	0	41	48	88	124
M-Satallax 1.4	100	94	0	91	122	136
M-Leo-II 1.2	24	14	4	0	50	93
GQML-Prover 1.2	34	28	9	24	0	81
f2p-MSPASS 3.0	44	42	1	45	59	0

based approach of f2p-MSPASS.[8] This problem set is too small to provide meaningful comparisons on the performance of f2p-MSPASS on first-order modal problems.

The classical leanTAP prover solves 283 problems (ignoring the modal operators) indicating that many problems are hard even for classical tableau-based ATP systems.

4 Conclusion

Despite the fact that modal logics are considered as one of the most important non-classical logics, the availability of *implementations* of automated theorem provers for *first-order* modal logics is very limited so far. In this paper several new ATP systems for various first-order modal logics based on different proof calculi and methods were introduced.

The performance of all new and existing ATP systems for first-order modal logic was evaluated on all 500 problems included in the first release v1.0 of the QMLTP library. Comprehensive statistics including different performance measures as well as illustrative performance graphs of the time complexity behavior were given for each considered ATP system.

Even though most of the 500 problems included in the first release of the QMLTP library have a rather syntactic nature, the QMLTP library serves as a useful basis in order to obtain a first impression of the performance of ATP systems for first-order modal logics.[9] Future releases of the QMLTP library will provide more problems from actual applications, thus, putting the testing and evaluation of ATP systems for modal logic on a more stable basis. All interested users are invited to submit new first-order problems to the QMLTP library.

An analysis of the performance results shows that the ATP system based on standard modal sequent calculi (performing an analytic tableau-like search) proves the highest number of problems. It slightly outperforms the system based on prefixed tableau calculi.[10] As for classical logic the ATP system based on an instance-based method finds by far the highest number of counter models. The systems using the embedding of modal logic into type theory show a solid performance as well. All considered ATP system solve most problems of the ones they solve at all within one second. This behavior is similar to that of classical ATP systems using standard tableau calculi. It indicates that the underlying proof calculi need to be improved.

The implementation of ATP systems for first-order modal logic is still in its infancy. Future work includes the implementation of, e.g., connection-based calculi for first-order modal logic and the extension of existing ATP systems to some first-order *multi*-modal logics.

Acknowledgements.

The authors would like to thank Christoph Benzmüller for providing the LEO-II system and the embedding into simple type theory, Ulrich Hustadt and Renate Schmidt for providing the MSPASS system, and Geoff Sutcliffe for help with the installation of some ATP systems and many useful suggestions.

[8]Of these problems ft2-MSPASS solves 16 problems (5 proved/11 counter models) for cumulative D, 15 problems (5/10) for constant D, 15 problems (9/6) for cumulative S4, and 9 problems (8/1) for constant S4.

[9]Observe the fact that problem libraries for *classical* first-order logic started with 75 (printed) problems in 1986. In 1997 the TPTP library v2.0.0 included the first 217 (non-clausal) *first-order* problems [41]. Its number has risen to over 5000 problems in the most recent version of the TPTP library.

[10]A similar behavior has already been shown by ATP systems for intuitionistic first-order logic [25].

References

[1] S. Autexier et al. VSE: formal methods meet industrial needs. *STTT*, 3(1):66–77, 2000.

[2] J. Backes, C. E. Brown. Analytic Tableaux for Higher-Order Logic with Choice. *IJCAR 2010*, LNCS 6173, pp. 76–90. Springer, 2010.

[3] P. Balsiger, A. Heuerding, S. Schwendimann. A Benchmark Method for the Propositional Modal Logics K, KT, S4. *Journal of Automated Reasoning*, 24:297–317, 2000.

[4] G. Beckert, R. Goré. Free Variable Tableaux for Propositional Modal Logics. In D. Galmiche, Ed., *TABLEAUX-1997*, LNAI 1227, pp. 91–106, Springer, 1997.

[5] B. Beckert, J. Posegga. leanTAP: Lean Tableau-based Deduction. *Journal of Automatic Reasoning*, 15(3): 339–358, 1995.

[6] C. Benzmüller, L. Paulson. Quantified Multimodal Logics in Simple Type Theory. Seki Report SR-2009-02 (ISSN 1437-4447), Saarland University, 2009.

[7] C. Benzmüller, L. Paulson, F. Theiss, A. Fietzke. LEO-II – A Cooperative Automatic Theorem Prover for Higher-Order Logic. *IJCAR 2008*, LNAI 5195, pp. 162–170. Springer, 2008.

[8] B. Beckert, M. Giese, R. Hähnle, V. Klebanov, P. Rümmer, S. Schlager, P. H. Schmitt. The KeY System (Deduction Component). In F. Pfennig, Ed., *CADE-21*, LNCS 4603, pp. 379–384. Springer, 2007.

[9] P. Blackburn, J. van Bentham, F. Wolter. *Handbook of Modal Logic*. Elsevier, 2006.

[10] V. Boeva, L. Ekenberg. A Transition Logic for Schemata Conflicts. *Data & Knowledge Engineering*, 51(3):277–294, 2004.

[11] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati. EQL-Lite: Effective First-Order Query Processing in Description Logics. In M. M. Veloso, Ed., *IJCAI-2007*, pp. 274–279, 2007.

[12] P. R. Cohen, H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[13] L. Fariñas del Cerro, A. Herzig, D. Longin, O. Rifi. Belief Reconstruction in Cooperative Dialogues. *AIMSA 1998*, LNCS 1480, pp. 254–266. Springer, 1998.

[14] M. Fitting. *Types, Tableaus, and Goedel's God*. Kluwer, 2002.

[15] M. Fitting, R. L. Mendelsohn. *First-Order Modal Logic*. Kluwer, 1998.

[16] G. Forbes. *Modern Logic. A Text in Elementary Symbolic Logic*. Oxford University Press, 1994.

[17] J. Garson. Quantification in Modal Logic. *Handbook of Philosophical Logic*, volume II, pp. 249–307. D. Reidel Publ. Co, 1984.

[18] R. Girle. *Modal Logics and Philosophy*. Acumen Publ., 2000.

[19] K. Gödel. An Interpretation of the Intuitionistic Sentential Logic. In J. Hintikka, Ed., *The Philosophy of Mathematics*, pp. 128–129. Oxford University Press, 1969.

[20] U. Hustadt, R. A. Schmidt. MSPASS: Modal Reasoning by Translation and First-Order Resolution. R. Dyckhoff., Ed., *TABLEAUX-2000*, LNAI 1847, pp. 67–81. Springer, 2000.

[21] F. Massacci, F. M. Donini: Design and Results of TANCS-2000 Non-classical (Modal) Systems Comparison. R. Dyckhoff., Ed., *TABLEAUX-2000*, LNAI 1847, pp. 50–56. Springer, 2000.

[22] C. Kreitz, J. Otten. Connection-based Theorem Proving in Classical and Non-classical Logics. *Journal of Universal Computer Science* 5, 88–112 (1999).

[23] R. C. Moore. A formal theory of knowledge and action. In J.R. Hobbs, R.C. Moore, Eds., *Formal Theories of the Commonsense World*, pp. 319–358. Ablex Pubs., 1985.

[24] J. Otten. ileanTAP: An Intuitionistic Theorem Prover. In D. Galmiche, Ed., *TABLEAUX-97*, LNAI 1227, pp. 307–312. Springer, 1997.

[25] J. Otten. leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In A. Armando, P. Baumgartner, G. Dowek, Eds., *IJCAR 2008*, LNCS 5195, S. 283–291. Springer, 2008.

[26] P. F. Patel-Schneider, R. Sebastiani. A New General Method to Generate Random Modal Formulae for Testing Decision Procedures. *Journal of Articial Intelligence Research*, 18:351–389, 2003.

[27] R. Petrick, F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *AIPS-2002*, pp. 212–221.AAAI Press, 2002.

[28] S. Popcorn. *First Steps in Modal Logic*. Cambridge University Press, 1994.

[29] T. Raths, J. Otten. The QMLTP Library: Benchmarking Theorem Provers for Modal Logics. Technical Report, University of Potsdam, 2011.

[30] T. Raths, J. Otten, C. Kreitz. The ILTP Problem Library for Intuitionistic Logic. *Journal of Automated Reasoning*, 38(1–3): 261–271, 2007.

[31] W. Reif, G. Schellhorn, K. Stenzel. Proving System Correctness with KIV 3.0. In W. McCune, Ed., *CADE-14*, LNAI 1249, pp. 69–72. Springer, 1997.

[32] R. Reiter. What Should a Database Know? *Journal of Logic Programming* 14(1–2):127–153, 1992.

[33] D. Sadek, P. Bretier, F. Panaget. ARTIMIS: Natural dialogue meets rational agency. In *IJCAI-1997*, pp. 1030–1035, 1997.

[34] S. Schulz. E - a brainiac theorem prover. *AI Communications*, 15(2):111–126, 2002.

[35] T. Sider. *Logic for Philosophy*. Oxford University Press, 2009.

[36] P.A. Spruit, R.J. Wieringa, J. Meyer. Regular database update logics. *TCS*, 254(1-2):591–661, 2002.

[37] M. Steedman, R. Petrick. Planning Dialog Actions. In S. Keizer, H. Bunt, T. Baek, Eds., *SIGdial 2007*, pp. 265–272, 2007.

[38] M. Stone. Abductive Planning With Sensing. In *AAAI-98*, pp. 631–636. Menlo Park CA., 1998.

[39] M. Stone. Towards a Computational Account of Knowledge, Action and Inference in Instructions. *Journal of Language and Computation*, 1:231–246, 2000.

[40] M. Stone, C. Doran, B. Webber, T. Bleam, M. Palmer. Microplanning with Communicative Intentions: The SPUD System. *Computational Intelligence*, 19(4):311–381, 2003.

[41] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[42] G. Sutcliffe. The CADE-22 automated theorem proving system competition – CASC-22.. *AI Communications*, 23(1):47–59, 2010.

[43] G. Sutcliffe. The 5th IJCAR automated theorem proving system competition – CASC-J5.. *AI Communications*, 24(1):75–89, 2011.

[44] V. Thion, S. Cerrito, M. Cialdea Mayer. A General Theorem Prover for Quantified Modal Logics. In U. Egly, C. G. Fermüller, Eds., *TABLEAUX-2002*, LNCS 2381, pp. 266–280. Springer, 2002.

[45] L. Wallen. *Automated deduction in nonclassical logic*. MIT Press, Cambridge, 1990.

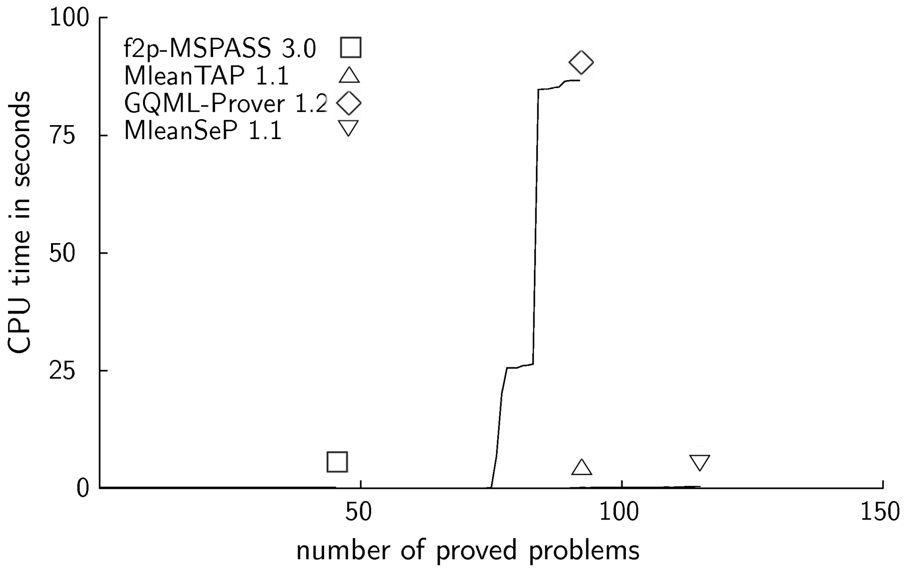

Figure 2: Performance graph for modal logic D with cumulative domains

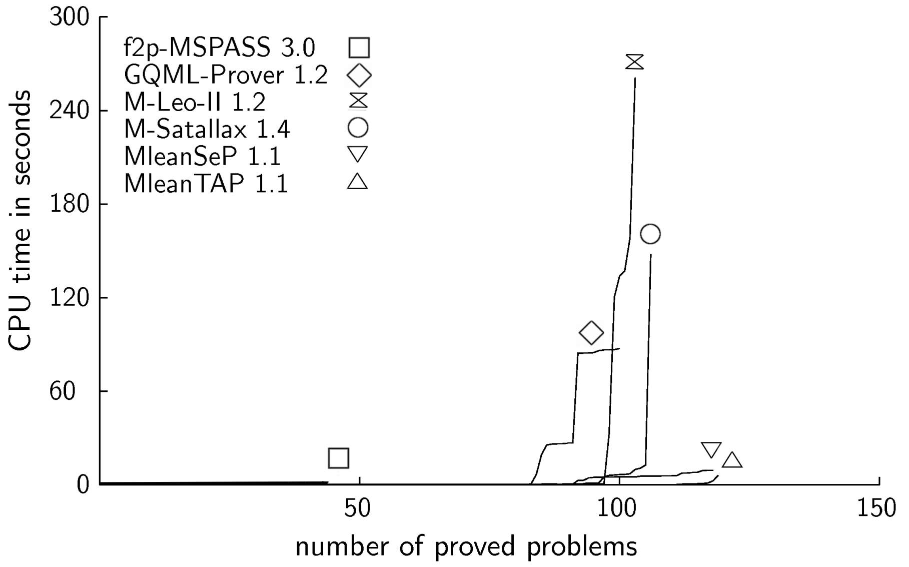

Figure 3: Performance graph for modal logic D with constant domains

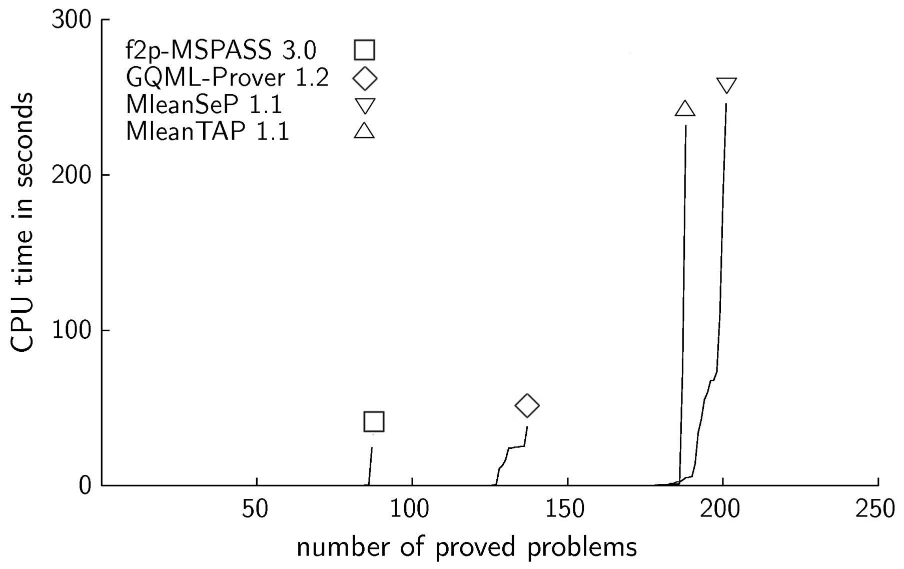

Figure 4: Performance graph for modal logic S4 with cumulative domains

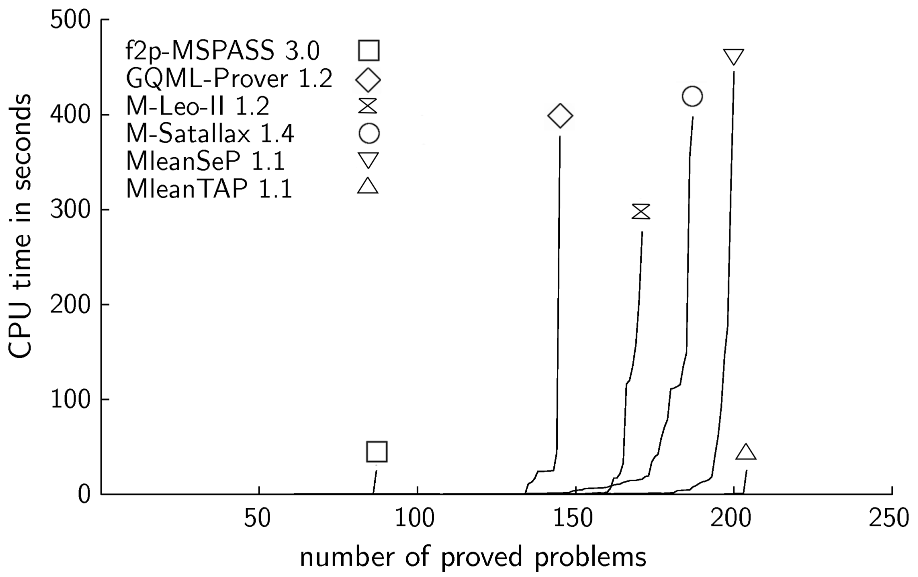

Figure 5: Performance graph for modal logic S4 with constant domains