Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 262 (2010) 113–125
www.elsevier.com/locate/entcs

Tableau Tool for Testing Satisfiability in LTL: Implementation and Experimental Analysis

Valentin Goranko[1]

DTU Informatics
Technical University of Denmark
Kgs. Lyngby, Denmark

Angelo Kyrilov[2] Dmitry Shkatov[3]

School of Computer Science
University of the Witwatersrand
Johannesburg, South Africa

Abstract

We report on the implementation and experimental analysis of an incremental multi-pass tableau-based procedure à la Wolper for testing satisfiability in the linear time temporal logic LTL, based on a breadth-first search strategy. We describe the implementation and discuss the performance of the tool on several series of pattern formulae, as well as on some random test sets, and compare its performance with an implementation of Schwendimann's one-pass tableaux by Widmann and Goré on several representative series of pattern formulae, including eventualities and safety patterns. Our experiments have established that Schwendimann's algorithm consistently, and sometimes dramatically, outperforms the incremental tableaux, despite the fact that the theoretical worst-case upper-bound of Schwendimann's algorithm, 2EXPTIME, is worse than that of Wolper's algorithm, which is EXPTIME. This shows, once again, that theoretically established worst-case complexity results do not always reflect truly the practical efficiency, at least when comparing decision procedures.

Keywords: LTL, satisfiability checking, incremental tableaux, implementation, one-pass tableaux.

1 Introduction

The multiple-pass incremental tableau-based decision procedure for the propositional linear-time logic LTL was first presented in print in [17]; the procedure builds on the ideas originally developed by Pratt for the propositional dynamic logic PDL in [11]. An analogous procedure was developed, at about the same time,

[1] Email: vfgo@imm.dtu.dk
[2] Email: angelo@cs.wits.ac.za
[3] Email: dmitry@cs.wits.ac.za

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2010.04.009

for the branching-time temporal logic UB by Ben-Ari, Manna, and Pnueli [2]. Subsequently, a number of other decision procedures based on the incremental tableau technology were developed, including our recent work [5,8] on the multi-agent epistemic logics with common and distributed knowledge, on temporal-epistemic logics [6,7], and the logics of strategic ability [4].

The one-pass tableau procedure was first developed for LTL by Schwendimann in [13,14] and recently applied to CTL by Abate, Goré, and Widmann in [1].

It is well-known that the worst-case complexity for LTL is PSPACE [16]. Unless applying on-the-fly pruning, however, the incremental tableau works in EXPTIME, while the worst-case complexity of Schwendimann's method is 2EXPTIME.

In this paper we report on the implementation and preliminary experimental analysis of an incremental tableau-based procedure à la Wolper for LTL. The implementation is available online at `http://msit.wits.ac.za/ltltableau`. We describe the implementation and discuss the performance of the tool on several series of pattern formulae, as well as on some random test sets, and compare its performance with the implementation of Schwendimann's one-pass tableau by Widmann and Goré on several typical series of pattern formulae. Our experiments have shown that Schwendimann's algorithm consistently, and sometimes dramatically, outperforms the multiple-pass incremental algorithm, despite the theoretical advantage of the latter. Schwendimann's algorithm even succeeds on some apparently difficult cases, on which reportedly (see [15], p.9-10) most automata-based tools fail to produce corresponding automata in a reasonable time and our multiple-pass tableaux-based tool fails to establish non-validity, too. We note that neither of the two implementations compared herein is aided by any special optimization techniques; thus, we essentially compare the two algorithms in their "pure" form. In particular, our tool implements a standard version of the algorithm from [17], based on a breadth-first search strategy, thus constructing the entire tableau before checking for existence of an open branch in it. Also, it should be taken into account that our tool was coded up in the Python programming language, while R. Goré and F. Widmann's tool was coded up in the OCaml language, which is known to be up to 100 faster than Python, and that should be taken into account when comparing the runtimes of the two implementations.

It is known, e.g., from research on description logics, that sometimes algorithms for theoretically computationally hard problems can solve most of the practically significant problems efficiently, especially when augmented with optimization techniques. Apparently, we face a similar phenomenon in the case of LTL as well, confirming that theoretical worst-case complexity results should be taken with a grain of salt when determining the practical utility of algorithms.

This paper, being a system description, only reports on the experimental performance comparison between the two tableau methods mentioned above. An in-depth theoretical analysis of the results will be presented in a follow-up work.

2 Preliminaries on the incremental multiple-pass tableaux for LTL

In this section, we briefly sketch out the incremental tableau procedure for LTL whose implementation is reported in this paper. We assume that the reader is familiar with the syntax and semantics of LTL (otherwise, see e.g., [17] or [3]).

In a nutshell, the incremental tableau procedure for testing an LTL-formula θ for satisfiability attempts to construct a graph $\mathcal{T}^\theta$, called a *tableau*, representing sufficiently many *Hintikka structures* for θ in the sense that if any Hintikka structure satisfies θ, then there is at least one represented by $\mathcal{T}^\theta$ that satisfies that formula. A Hintikka structure for θ is, essentially, a finite partial representation of a model for θ. It is not hard to prove that an LTL formula is satisfiable in a model iff it is satisfiable in a Hintikka structure (for details see, e.g., [6]). Thus, an LTL-formula θ is satisfiable iff the procedure for θ succeeds.

The tableau procedure consists of two major phases: *construction*, and *elimination*, the latter, in turn, consisting of *pre-state elimination*, and *state elimination*.

During the construction phase, a directed graph $\mathcal{P}^\theta$—referred to as the *pretableau* for θ—is produced. Its set of nodes properly contains the set of nodes of the tableau $\mathcal{T}^\theta$ that the procedure is ultimately trying to build. Nodes of $\mathcal{P}^\theta$ are sets of LTL-formulae, some of which—referred to as *states*— represent states of a Hintikka structure (and, therefore, states of a model), while others—referred to as *pre-states*—fulfill a technical role, in particular of helping to keep $\mathcal{P}^\theta$ finite.

During the pre-state elimination phase, a smaller graph $\mathcal{T}_0^\theta$ is created out of $\mathcal{P}^\theta$—referred to as the *initial tableau for* θ—by eliminating all the pre-states from $\mathcal{P}^\theta$, as they have already fulfilled their role, and redirecting the edges.

Lastly, during the state elimination phase, all, if any, states of $\mathcal{T}_0^\theta$ are removed that cannot be satisfied in a Hintikka structure, for one of the following reasons: either they are *patently inconsistent*, i.e., contain a complementary pair of formulae $\psi, \neg\psi$, or contain unrealizable eventualities (i.e., formulae of the form $\varphi \mathcal{U} \psi$ such that no state containing ψ can be reached along the states containing φ from the state in question), or do not have any successors (which is against the LTL-semantics), e.g, because all their successors may have been eliminated earlier.

Note, that the removal of "bad" states may have to be repeated many times until a stable configuration is reached, hence the term "multiple-pass" tableau.

The result of the overall procedure is a (possibly empty) subgraph $\mathcal{T}^\theta$ of $\mathcal{T}_0^\theta$, referred to as the *final tableau for* θ. Then, if there is some state Δ in $\mathcal{T}^\theta$ containing θ, the procedure pronounces θ satisfiable; otherwise, θ is declared unsatisfiable.

The completeness proof shows how to build a Hintikka structure, and thus, a model, out of a non-empty final tableau, while the soundness proof shows that the final tableau for any unsatisfiable formula will always be empty.

We will describe briefly the three stages mentioned above in the next section, while describing the implementation. Before that, we need to introduce some standard terminology and notation that will be used later on.

The tables below list the types of LTL formulae classified as α's (conjunctions)

and β's (disjunctions), together with their respective conjuncts and disjuncts.

α	α_1	α_2	β	β_1	β_2
$\neg\neg\varphi$	φ	φ	$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\varphi \wedge \psi$	φ	ψ	$\varphi \vee \psi$	φ	ψ
$\neg(\varphi \vee \psi)$	$\neg\varphi$	$\neg\psi$	$(\varphi \mathcal{U} \psi)$	ψ	$\varphi \wedge \mathcal{X}(\varphi \mathcal{U} \psi)$
$\neg\mathcal{X}\varphi$	$\mathcal{X}\neg\varphi$	$\mathcal{X}\neg\varphi$	$\neg(\varphi \mathcal{U} \psi)$	$\neg\psi \wedge \neg\varphi$	$\neg\psi \wedge \neg\mathcal{X}(\varphi \mathcal{U} \psi)$
$\mathcal{G}\varphi$	φ	$\mathcal{X}\mathcal{G}\varphi$	$\neg\mathcal{G}\varphi$	$\neg\varphi$	$\neg\mathcal{X}\mathcal{G}\varphi$

All the other formulae (propositional parameters and constants, as well as the formulae of the form $\mathcal{X}\varphi$) are called *primitive*. Unlike the case of α- and β-formulae, their truth at a state of a model cannot be reduced to the truth of simpler formulae *at the same state*. A set of LTL-formulae Σ is said to be *downwards-saturated* if, first, $\alpha \in \Sigma$ implies that both $\alpha_1 \in \Sigma$ and $\alpha_2 \in \Sigma$, and second, if $\beta \in \Sigma$ implies that either $\beta_1 \in \Sigma$ or $\beta_2 \in \Sigma$. A set of formulae Δ is a *maximal downward saturated extension* of the set Γ if, first, Δ is downward-saturated, and second, there is no downward-saturated Δ' such that $\Gamma \subseteq \Delta' \subset \Delta$.

3 Description of the implementation

3.1 Syntax

The algorithm takes as input the formula to be tested (represented by a string), and returns the string `'satisfiable'` if the formula is found to be satisfiable or, otherwise, `'not satisfiable'`. The implementation supports all the usual Boolean and temporal connectives. These are `A` for $\wedge$, `O` for $\vee$, `I` for $\rightarrow$, `N` for $\neg$, `U` for 'Until', `F` for 'Sometime in the future', `G` for 'Always in the future', and `X` for 'Nexttime'. The formulae are inductively defined as follows:

(i) Every propositional variable, encoded here by lower-case Latin letter followed by a decimal, such as `a12`, is a formula.
(ii) If φ is a formula then `N`φ, `X`φ, `F`φ and `G`φ are formulae.
(iii) If φ and ψ are formulae then `(`φ `A` ψ`)`, `(`φ `O` ψ`)`, `(`φ `I` ψ`)` and `(`φ `U` ψ`)` are formulae.

3.2 Data Structures

The tableau is a directed graph, made up of states and pre-states. The generic term *node* will be used to refer to either states or pre-states when it is not important to distinguish between the two. The graph is implemented as a list of nodes. Each node is a record that contains the following fields:

`id`: A unique integer identifier for the node.
`parents`: A list of integers containing the ids of the parents of the node.
`children`: A list of integers containing the ids of the children of the node.

`type`: A string that specifies what type the node is. Possible values are `pre`, `proto` and `state`.
`formulae`: A list of strings containing all formulae that are true at the given node.
`marked`: A Boolean flag used for checking eventualities.
`succMarked`: A Boolean flag showing whether successor nodes are marked.

3.2.1 Construction Phase

As already explained, the construction phase produces a graph containing two kinds of node , states and pre-states. Technically, states, unlike pre-states, are required to be downward-saturated (see above). The graph also contains two kinds of edge. One kind of edge connects pre-states to states, and is denoted here by the double arrow $\Rightarrow$. The other kind of edge connects states to pre-states, and is denoted here by the single arrow $\longrightarrow$ (proto-states mentioned above are part of the implementation, but do not feature in a high-level description of the procedure; essentially, they are "states in the making"). The construction procedure for a formula θ begins with creating a single pre-state $\{\theta\}$. Afterwards, the procedure alternates between creating states from pre-states using rule **SR** stated below, and pre-states from states using rule **PR** stated below, until we reach saturation.

SR Given a pre-state Γ to which **SR** has not been applied yet, do the following:

(i) add all maximal downward-saturated extensions of Γ that are not patently inconsistent to the pretableau as *states*;
(ii) for each of the newly added states Δ, if Δ does not contain formulae of the form $\mathcal{X}\varphi$, add $\mathcal{X}\top$ to it; call the result Δ';
(iii) for each so created Δ', put $\Gamma \Rightarrow \Delta'$;
(iv) if, however, the part of the pretableau constructed so far already contains Δ', do not create a new copy of Δ', but simply put $\Gamma \Rightarrow \Delta'$.

PR Given a state Δ to which **PR** has not been applied yet, do the following:

(i) add to the pretableau the set of the form $\Gamma = \{\varphi \mid \mathcal{X}\varphi \in \Delta\}$ as *pre-state*, provided it is not patently inconsistent;
(ii) for each so created Γ, put $\Delta \longrightarrow \Gamma$;
(iii) if, however, the pretableau already contains Γ, do not create a new copy of Γ, but simply put $\Delta \longrightarrow \Gamma$.

In the implementation, the construction algorithm starts off by creating the initial node of the tableau. This is the pre-state labelled with the input formula. The two construction rules are then applied continuously until no new nodes are added. The two construction methods, corresponding to the rules described above, are called *alphaBetaRules* and *nextTimeRule*. The *alphaBetaRules* method creates states from pre-states (the intermediate results are called proto-states) by a process of downward saturation and the *nextTime* method creates pre-states from states.

3.2.2 Elimination Phase

The elimination phase begins by removing all the pre-states and all the $\Rightarrow$ edges from the pre-tableau, and accordingly redirecting $\longrightarrow$ edges. The result is called the *initial tableau*. After that, we start eliminating "bad" states. Recall that these are states that are inconsistent, states that contain unfulfilled eventualities, and states that have no successors. The removal of prestates and inconsistent states is trivial. A naive way of checking whether eventualities have been fulfilled may cause the algorithm to run for extremely long time, so a more efficient ranking procedure, called *removeEventualities* is used to detect unfulfilled eventualities. It begins by finding all eventualities in the tableau and storing them in a list. For each eventuality in the list the algorithm does the following:

(i) For every state, set *marked* to false.

(ii) Find all states that fulfill that eventuality and set for them *marked* to true.

(iii) Mark all states whose successors are marked.

(iv) Repeat step (iii) until no more states can be marked.

(v) Remove all states that contain the eventuality and have not been marked.

The *removeNonSuccessors* procedure looks for states with no successors and removes them. The *removeEventualities* and *removeNonSuccessors* procedures are applied repeatedly until no more states can be removed from the tableau. The result is the *final tableau* structure.

The last step is to check if the final tableau is open or closed. To do this we check if the tableau contains any of its initial states. These are states that contain the input formula. If the tableau is found to be open then the algorithm returns `'satisfiable'` otherwise it returns `'not satisfiable'`.

3.3 The Tableau Algorithm

The main tableau algorithm, as described above, is shown in figure 1 below.

4 Testing and analysis

4.1 Correctness testing

A large set of random formulae was generated for the empirical testing of the correctness of the implementation. These were tested on our as well as other available tools, in particular, on R. Goré and F. Widmann's implementation of Schwendimann's one-pass tableau, also chosen for the performance comparison. After removing bugs in earlier versions, both tools returned consistent results on all tests. A comparison of the running times is presented in the rest of the paper. All tests were conducted on an Intel Xeon 8-core architecture with 8 GB RAM and the Mac OS X v10.5 operating system. Our tool was coded up in the Python programming language, while R. Goré and F. Widmann's tool was coded up in the Ocaml lan-

```
Test(formula) {
    tableau = constructPretableau(formula)
    tableau = removepre-states(tableau)
    tableau = removeInconsistent(tableau)

    while(True) {
        current = tableau;
        tableau = removeEventualities(tableau)
        tableau = removeNonSuccessors(tableau)

        if(current == tableau){
            break
        }
    }
    return isOpen(tableau)
}
```

Fig. 1. The incremental tableau algorithm for testing satisfiability of LTL formulae

guage. [4]. Memory usage by both tools was carefully monitored during all tests to ensure that it does not run out. That would cause the computer to start using virtual memory and greatly increase the running times. Virtual memory was not used in any of the tests reported below.

Another way to test the correctness of the implementation was to generate and test large sets of formulae that we knew to be satisfiable, and formulae we knew to be not satisfiable. In particular, we have used sets of such formulae generated by M. Montali [10], and one of them detected a bug in an earlier version of the program.

4.2 Pattern Series

The first set of tests conducted were on pattern series. The patterns we used were taken from the paper of Rozier and Vardi [12], used there to test the performance of automata based tools for testing satisfiability of LTL formulae.

The diagrams in this section present the running times of the two tableau tools on the different patterns, to which we will refer hereafter as 'Wolper's tableau' and 'Schwendimann's tableau', because both implementations faithfully represent the respective algorithms, without any special features that may slow down or speed up the performance in specific cases.

For running times of automata-based tools on the patterns presented below, the reader is referred to Rozier and Vardi [12]. Later on, we briefly discuss the performance comparisons between the automata-based tools and the tableau tool presented here.

The first pattern is the *E-formulae* pattern, being a conjunction of eventualities,

[4] A precise ratio between the performance speeds of the two languages is impossible to determine, as such a ratio depends on a particular computational task, but it is known that Ocaml can be up to 100 faster than Python, and that should be taken into account when comparing the runtimes of the two implementations.

of the form $\mathcal{F}p1 \wedge \mathcal{F}p2 \wedge \ldots \wedge \mathcal{F}pn$. The pattern was tested on input sizes varying from $n = 1$ to $n = 10$. The running times are given on the graph in figure 2.

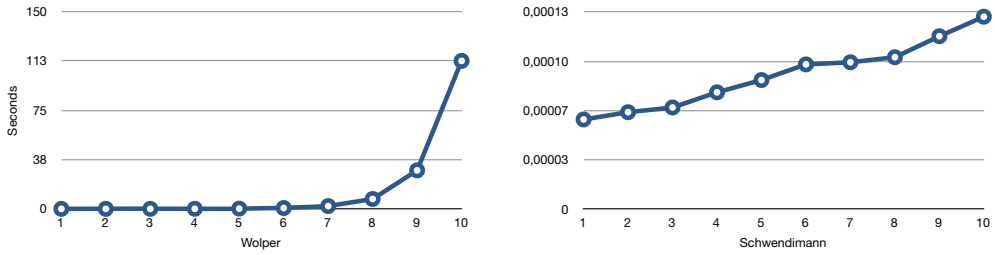

Fig. 2. Running time of E formulae

Wolper's tableau is not able to verify E formulae of more than 10 conjuncts within a reasonable time. As the input grows beyond $n = 7$, we see an exponentially sharp increase in running time. This increase is caused by the procedure that checks whether eventualities are realized. For example when $n = 10$ the program generates over 120 000 nodes in the tableau and 10 eventualities have to be checked. On the other hand, the running time of Schwendimann's tableau grows linearly because there is no separate procedure for checking eventualities.

The next pattern tested was the *S-formulae* pattern, of the form of $\mathcal{G}p1 \wedge \mathcal{G}p2 \wedge \ldots \wedge \mathcal{G}pn$. There are no eventualities in this formula pattern, so we should expect much better results, compared to the E-formulae patter. Indeed, the graph in figure 3 confirms this expectation.

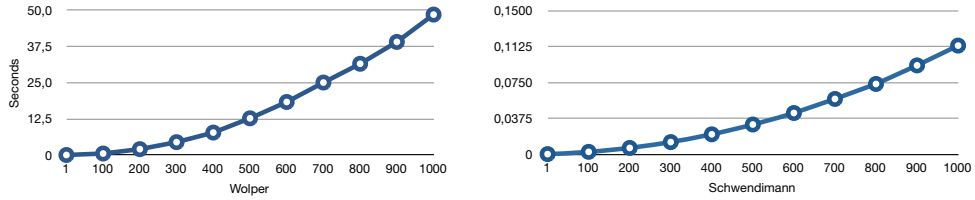

Fig. 3. Running time of S formulae

Both algorithms perform on this pattern series in a quadratic time, because these formulae involve no branching and no eventualities. The running time of Wolper's tableau is dominated by the procedure of removing pre-states, which is a costly procedure especially in highly connected graphs. However, the difference between the running times of the two algorithms is only a constant factor.

The next two patterns involve nested *Until* operators. The first of them, the U_1*-formulae* pattern, is nesting in the first argument: $(((p1\ \mathcal{U}\ p2)\ \mathcal{U}\ p3)\ \mathcal{U}\ \ldots pn)$. The running times of both algorithms are shown in figure 4.

Again, Wolper's tableau only manages to verify formulae with a very low n value. That is because, for a formula of size n, there are n eventualities to be checked.

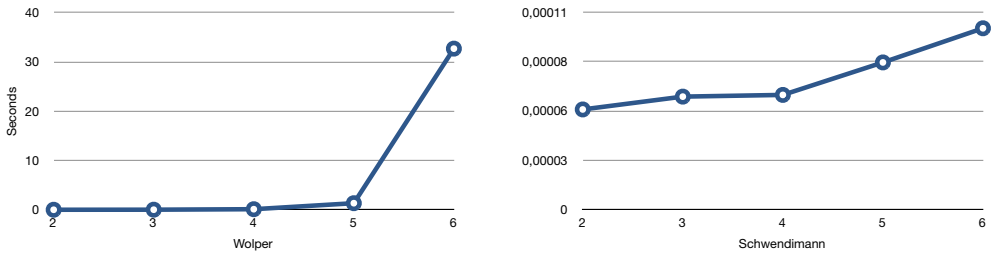

Fig. 4. Running time of U_1 formulae

Also for $n = 7$ the program generates over 68 000 nodes. Again, Schwendimann's tableaux show vastly better behaviour here.

The U_2-*formulae* pattern has nesting on the second argument of *Until*, of the form $(p1\ \mathcal{U}\ (p2\ \mathcal{U}\ (p3\ \mathcal{U} \ldots pn))$. Formulae of this pattern contain n eventualities, too, but Wolper's tableau generates very few states compared to the U_1-formulae pattern. That is why the algorithm manages to verify much larger input formulae. The running times are shown in figure 5. Schwendimann's tableaux perform better again, but as can be seen from the graph, its running time curve grows at a similar rate to that for Wolper's algorithm.

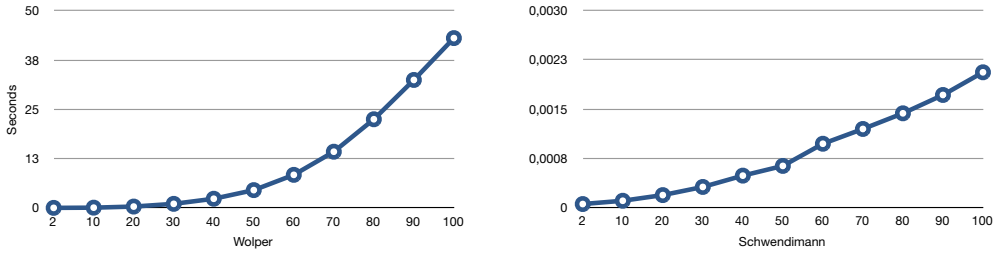

Fig. 5. Running time of U_2 formulae

The last pattern sets are the so called C-*formulae* patterns. They are made up of subformulae of the form $\mathcal{GF}pi$. The pattern C_1 is a disjunction of such subformulae, and C_2 is a conjunction of such subformulae. The running times of the algorithms on C_1-formulae are shown in figure 6.

Wolper's tableau succeeds to verify reasonably-sized formulae of the C_1-formulae pattern because very few nodes are generated. The small number of states allows the procedure to check all n eventualities in a reasonable time. The running time of Schwendimann's tableau grows at a similar rate but again with a much lower constant factor.

The C_2 pattern is a conjunction of the form $\mathcal{GF}p1 \wedge \mathcal{GF}p2 \wedge \ldots \wedge \mathcal{GF}pn$. The running time of the algorithm on C_2 formulae is shown in figure 7.

The running time of Wolper's tableau increases sharply after $n = 7$ because the program begins to generate exponentially many nodes. The need to check whether

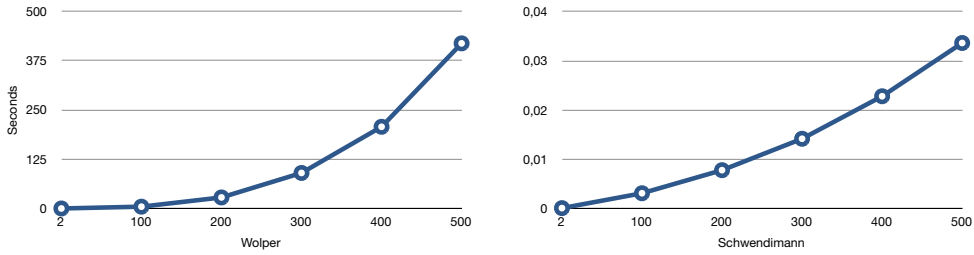

Fig. 6. Running time of C_1 formulae

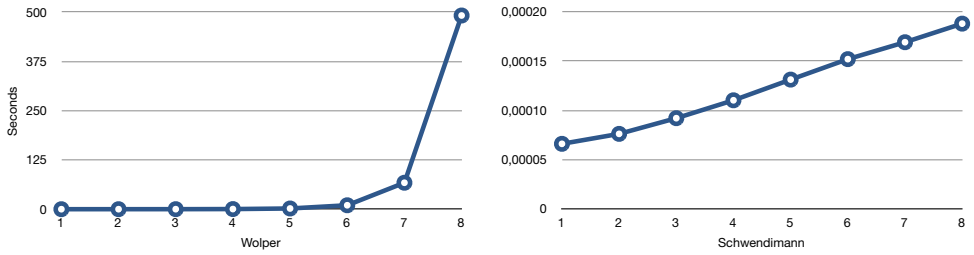

Fig. 7. Running time of C_2 formulae

a lot of eventualities are fulfilled, together with the high number of states, results in poor performance. Schwendimann's tableaux, where there is no elimination procedure, run in linear time for this formula pattern, too.

Other pattern series used to compare Wolper's tableau to Schwendimann's tableau were generated by M. Montalli [10]. These patterns use two parameters n, d, shown on the abscissa of the graphs on Fig. 8 and 9, where the running times for both tools are plotted. The first parameter is the number of propositional variables and the second is the nesting depth of temporal patterns. For instance, in one of the series the formula $\mathcal{F}(a_1 \wedge \mathcal{X}\mathcal{F}(a_1 \wedge \mathcal{X}\mathcal{F}a_1))$ has parameters $(1, 2)$, meaning that it contains 1 variable and the pattern $\mathcal{X}\mathcal{F}$ has a nesting depth 2.

For a description of the other patterns and further details on them, see [9].

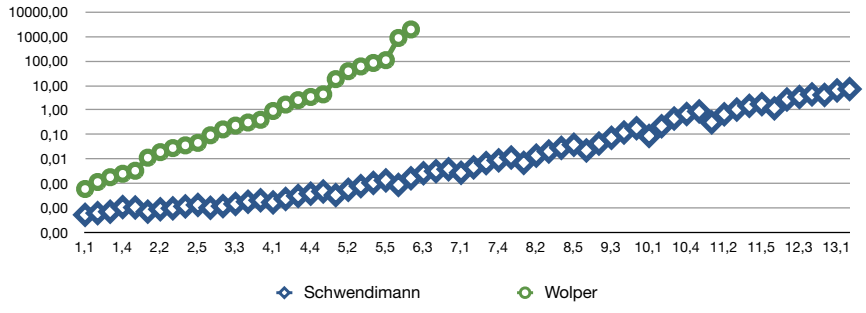

Fig. 8. Running time of Montali's satisfiable formulae

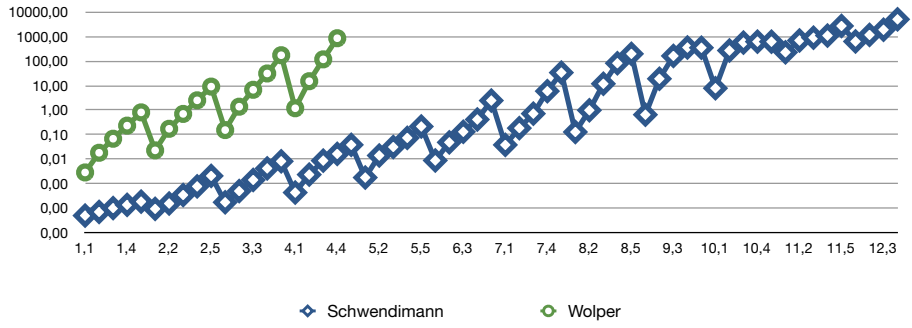

Fig. 9. Running time of Montali's unsatisfiable formulae

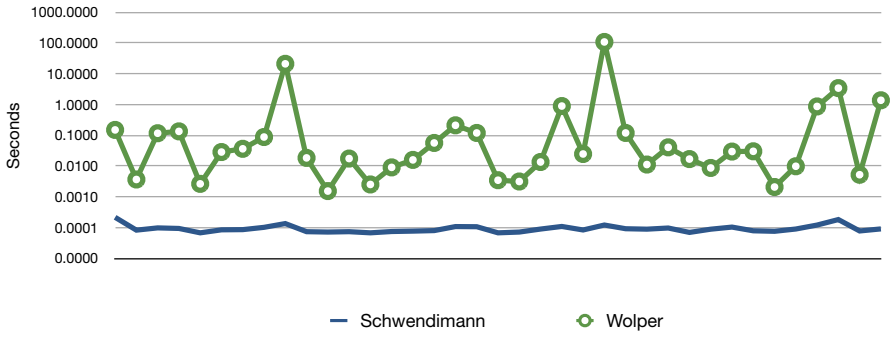

Fig. 10. Running time of random formulae

4.3 Random formulae

A random formula generator was used to generate random test formulae of different sizes. The parameters used were: n – the number of propositional variables, and d – the nesting depth for operators. Wolper's tableau manages to verify formulae with low nesting depth in a very reasonable time. When the depth is increased to 5 and beyond, the algorithm begins to struggle. As we can see from the graph in figure 10, which shows random formulae of two variables and nesting depth of 5, certain formulae go well beyond the 0.5 second mark. These are all the spikes in the graph, some of which reach times of over 100 seconds. For random formulae of more than 6 propositional variables and nesting depth over 5, Wolper's tableau has running times of over 1000 seconds while Schwendimann's tool is consistently fast.

4.4 Performance comparisons with automata-based tools

In their paper, Rozier and Vardi tested both explicit and symbolic automata-based tools for LTL satisfiability checking. The implementation of Wolper's tableau compares well with the explicit tools, but is not as efficient as the symbolic ones. On the other hand, Schwendimann's tableaux have proved to be much more efficient on some formulae patterns.

4.5 Summary of results

The purpose of doing the experimental analysis reported in this paper was to verify the correctness of the implementation, to test the performance, and to compare it with the performance of Schwendimann's tableaux. The results of the performance testing can be used to determine the suitability of this tool for industrial use, at least for specific formulae patterns.

The correctness was successfully verified with practical certainty, as the last version of the implementation of Wolper's tableau returned correct answers for all the formulae that were tested on it. Also the individual sub-procedures of the tableau were tested independently to ensure their correctness.

As for performance, for formula patterns with no eventualities to be checked, the running times of Wolper's tableau and Schwendimann's tableaux grow at the same rate, typically the growth of the running time of Schwendimann's tableau having much lower constant factors. However, for the formula patterns described above that cause generation of many nodes and there are many eventualities, the running time of Wolper's tableau grows exponentially on the input size, whereas Schwendimann's remains linear.

5 Concluding remarks and future work

In future work, we intend to analyze and compare theoretically the incremental multiple-pass, and the one-pass tableau methods and to provide a theoretical explanation of the superior performance of the latter, while identifying the scope of that superior performance and indicating the cases where the multi-pass tableaux perform better. We are also planning to investigate optimization techniques of both methods. In particular, we intend to implement a modified version of Wolper's algorithms based on a depth-first, as opposed to breadth-first, search strategy, which will not create all offspring states of a given prestate, but only as many as necessary to realize all eventualities passed from the predecessor states. The ultimate goal of studying such optimization techniques is to design a "hybrid" procedure using the most optimal features of the both tableau procedures considered in this paper as well as optimization techniques.

6 Acknowledgments

This work is based on the Masters project of the second author (A. Kyrilov), supervised by the other two authors. It has been partly supported by the National Research Foundation of South Africa.

We are indebted to Rajeev Goré and Florian Widmann for providing us with their implementation of Schwendimann's one-pass procedure and for many discussions on its merits – and this paper can be regarded, inter alia, as a vindication of Rajeev's sustained, but largely shrugged off by the modal logic community, claims on the superior practical performance of Schwendimann's one-pass tableau method.

We are also grateful to Marco Montali for testing and finding a bug in an earlier

version of out tool, and for providing us with his benchmark formulae series. Finally, we thank the anonymous referees for their constructive and encouraging remarks.

References

[1] Pietro Abate, Rajeev Goré, and Florian Widmann. One-pass tableaux for Computation Tree Logic. In Nachum Dershowitz and Andrei Voronkov, editors, *Proceedings of LPAR 07*, volume 4790 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 2007.

[2] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. In *Proc. 8th ACM Symposium on Principles of Programming Languages, also appeared in Acta Informatica, 20(1983), 207-226*, pages 164–176, 1981.

[3] E. Allen Emerson. Temporal and modal logics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. MIT Press, 1990.

[4] Valentin Goranko and Dmitry Shkatov. Tableau-based decision procedures for logics of strategic ability in multi-agent systems. To appear in *ACM Transactions on Computational Logic*. Available at `http://tocl.acm.org/accepted.html`.

[5] Valentin Goranko and Dmitry Shkatov. Tableau-based decision procedure for the multi-agent epistemic logic with operators of common and distributed knowledge. In Antonio Cerone and Stefan Gunter, editors, *Proceedings of the sixth IEEE Conference on Sofware Engineering and Formal Methods (SEFM 2008)*, pages 237–246. IEEE Computer Society Press, 2008.

[6] Valentin Goranko and Dmitry Shkatov. Tableau-based decision procedure for full coalitional multiagent temporal-epistemic logic of linear time. In Decker, Sichman, Sierra, and Castelfranchi, editors, *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), May 2009, Budapest, Hungary*, 2009.

[7] Valentin Goranko and Dmitry Shkatov. Tableau-based decision procedure for the full coalitional multiagent temporal-epistemic logic of branching time. In *to appear in: Proc. of the Workshop on Formal Approaches to Multi-Agent Systems FAMAS'09, 7-11 September 2009, Torino, Italy*, 2009.

[8] Valentin Goranko and Dmitry Shkatov. Tableau-based procedure for deciding satisfiability in the full coalitional multiagent epistemic logic. In Sergei Artemov and Anil Nerode, editors, *Proc. of the Symposium on Logical Foundations of Computer Science (LFCS 2009)*, volume 5407 of *Lecture Notes in Computer Science*, pages 197–213. Springer-Verlag, 2009.

[9] M. Montali, P. Torroni, M. Alberti, F. Chesani, E. Lamma, and P. Mello. Abductive logic programming as an effective technology for the static verification of declarative business processes. *J. of Algorithms in Cognition, Informatics and Logic*, page to appear, 2009.

[10] Marco Montali. personal communication. 2009.

[11] Vaughan R. Pratt. A near optimal method for reasoning about action. *Journal of Computer and System Sciences*, 20:231–254, 1980.

[12] K.Y. Rozier and M.Y. Vardi. LTL satisfiability checking. In *14th Workshop on Model Checking Software (SPIN '07)*, volume 4595 of *Lecture Notes in Computer Science (LNCS)*, pages 149–167. Springer-Verlag, 2007.

[13] S. Schwendimann. *Aspects of Computational Logic*. PhD thesis, Universität Bern, Switzerland, 1998.

[14] Stefan Schwendimann. A new one-pass tableau calculus for pltl. In H. de Swart, editor, *Proceedings of TABLEAUX'98*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 277–291. Springer-Verlag, 1998.

[15] Roberto Sebastiani, Stefano Tonetta, and Moshe Y. Vardi. Symbolic systems, explicit properties: On hybrid approaches for ltl symbolic model checking. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 350–363. Springer, 2005.

[16] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. In *STOC*, pages 159–168. ACM, 1982.

[17] Pierre Wolper. The tableau method for temporal logic: an overview. *Logique et Analyse*, 28(110–111):119–136, 1985.