TR 26/98 University of Karlsruhe

Combining Local and Global Model Checking *

Armin Biere[1] and Edmund M. Clarke[2] and and Yunshan Zhu[2]

[1] Universität Karlsruhe, Fakultät für Informatik,
Postfach 6980, 76128 Karlsruhe
[2] Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A
Edmund.Clarke@cs.cmu.edu

Abstract. The verification process of reactive systems in *local model checking* [2, 9, 28] and in *explicit state model checking* [14, 16] is *on-the-fly*. Therefore only those states of a system have to be traversed that are necessary to prove a property. In addition, if the property does not hold, than often only a small subset of the state space has to be traversed to produce a counterexample. *Global model checking* [8, 24] and, in particular, *symbolic model checking* [6, 23] can utilize compact representations of the state space, e.g. BDDs [5], to handle much larger designs than what is possible with local and explicit model checking. We present a new model checking algorithm for LTL that combines both approaches. In essence, it is a generalization of the tableau construction of [2] that enables the use of BDDs but still is on-the-fly.

1 Introduction

Model Checking [8, 24] is a powerful technique for the verification of reactive systems. With the invention of symbolic model checking [6, 23] very large systems, with more than 10^{20} states, could be verified. However, it is often observed, that explicit state model checkers [11] outperform symbolic model checkers, especially in the application domain of asynchronous systems and communication protocols [12]. We believe that the main reasons are the following: First, symbolic model checkers traditionally use binary decision diagrams (BDDs) [5] as an underlying data structure. BDDs trade space for time and often their sheer size explodes. Second, depth first search (DFS) is used in explicit state model checking, while symbolic model checking usually traverses the state space in breadth first search (BFS). DFS helps to reduce the space requirements and is able to find counterexamples much faster. Finally, global model checking traverses the state space backwards, and can, in general, not avoid visiting non reachable states without a prior reachability analysis.

In [3] a solution to the first problem, and partially to the second problem, was presented, by replacing BDDs by SAT (propositional satisfiability checking procedures). In this paper we propose a solution to the second and third problems of symbolic model checking. Our main contribution is a new model checking algorithm that generalizes the tableau construction [2] of local model checking for LTL and enables the use of BDDs. It is based on a mixed DFS and BFS strategy and traverses the state space in a forward oriented manner.

Our research is motivated by the success of forward model checking [17, 18]. Forward model checking is a variant of symbolic model checking in which only forward image computations are used. Thus it mimics the *on-the-fly* nature of explicit and local model checking in visiting only reachable states. Note that [18] presented a technique for the combination of the BFS, used in BDD based approaches, with the DFS of explicit state model checkers. It was shown that especially this feature enables forward model checking to find counterexamples much faster. However, only a restricted class of properties, i.e. path expressions, can be handled by the algorithms of [17, 18].

* This research is sponsored by the Semiconductor Research Corporation (SRC) under Contract No. 98-DJ-294, and the National Science Foundation (NSF) under Grant No. CCR-9505472. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of SRC, NSF, or the United States Government.

Henzinger et. al. in [15] partially filled this gap by proving that all properties specified by Büchi Automata, or equivalently all ω-regular properties, can be processed by forward model checking. In particular, they define a forward oriented version of the modal μ-calculus [20], called *post-μ*, and translate the model checking problem of a ω-regular property into a *post-μ* model checking problem. Because LTL (linear temporal logic) properties can be formulated as ω-regular properties [29], their result implies that all LTL properties can be checked by forward model checking.

The fact, that LTL can be checked by forward model checking, can also be derived by applying the techniques of [17] to the tableau construction of [7]. However, this construction and also [15] do not allow the mixture of DFS and BFS, as in the layered approach of [18]. In addition, DFS was identified as a major reason that explicit state model checking is able to outperform symbolic model checking on certain examples.

The contribution of our paper is the following. First we present a new model checking algorithm that operates directly on LTL formulae. For example [15] requires two translations, from LTL to Büchi Automata and then to *post-μ*. A similar argument applies to [7, 10]. Second it connects the local model checking paradigm of [2] with symbolic model checking in a natural way, thus combining BDD based with on-the-fly model checking. For the modal μ-calculus this connection has already been made in [4]. However, a direct application of [4] to the tableau construction of [2], our multiple state tableau construction, results in a tableau that is exponential in the size of the model. Only the introduction of a split rule in combination with efficient splitting heuristics allows us to keep the tableau linear in the size of the model. Finally our approach shows, that the idea of mixing DFS with BFS can be lifted from path expressions [18] to LTL.

Our paper is organized as follows. In the next section our notation is introduced. Section 3 presents a variation of the single state tableau construction of [2], on which our multiple state tableau construction, introduced in Section 4, is based. The following section discusses implementation details of the algorithm. In Section 6 we investigate heuristics to generate small tableau. An important optimization is presented in Section 7. The technical part ends with a discussion of the complexity and comparison with related work in Section 8. Finally we address open issues.

2 Preliminaries

A *Kripke structure* is a tuple $K = (\Sigma, \Sigma_0, \delta, \ell)$ with Σ a finite set of states, $\Sigma_0 \subseteq \Sigma$ the set of initial states, $\delta \subseteq \Sigma \times \Sigma$ the transition relation between states, and $\ell : \Sigma \to \mathbb{P}(\mathcal{A})$ the labeling of the states with *atomic propositions* $\mathcal{A} = \{p, \ldots\}$. For technical reasons we assume that every state has at least one successor state. A *path* $\pi = (s_0, s_1, \ldots)$ is an infinite sequence of states $s_i \in \Sigma$. Define $\pi(i) = s_i$ as the i-th state in π. We also use the notation $\pi^i = (\pi(i), \pi(i+1), \ldots)$ for the suffix of π starting at $\pi(i)$. The image operation on a set of states $S \subseteq \Sigma$ is defined as $\mathrm{Img}(S) := \{t \in \Sigma \mid \exists s \in S.\ (s,t) \in \delta\}$.

As temporal operators we consider, the *next time* operator **X**, the *finally* operator **F**, the globally operator **G**, the *until* operator **U**, and its dual, the *release* operator **R**. An LTL formula f is called an *eventuality* iff $f = \mathbf{F}h$ or $f = (g\,\mathbf{U}\,h)$. In this case h is called the *body* of f. We assume the formulae to be in negation normal form, as in [2, 9, 10]. Thus negations only occur in front of atomic propositions. This restriction does not lead to an exponential blow up because we have included the **R** operator that fulfills the property $\neg(f\,\mathbf{U}\,g) \equiv \neg f\,\mathbf{R}\,\neg g$.

An LTL formulae f holds on a path π, written $\pi \models f$, according to the following definitions:

$$
\begin{array}{lll@{\qquad}lll}
\pi \models p & \textit{iff} & p \in \ell(\pi(0)) & \pi \models \neg p & \textit{iff} & p \notin \ell(\pi(0)) \\
\pi \models g \wedge h & \textit{iff} & \pi \models g \text{ and } \pi \models h & \pi \models g \vee h & \textit{iff} & \pi \models g \text{ or } \pi \models h \\
\pi \models \mathbf{G}g & \textit{iff} & \forall i.\, \pi^i \models g & \pi \models \mathbf{F}h & \textit{iff} & \exists i.\, \pi^i \models h \\
\pi \models \mathbf{X}g & \textit{iff} & \pi^1 \models g & & & \\
\pi \models g\,\mathbf{U}\,h & \textit{iff} & \exists i.\, \pi^i \models h \text{ and } \forall j < i.\, \pi^j \models g & & & \\
\pi \models g\,\mathbf{R}\,h & \textit{iff} & \forall i.\, \pi^i \models h \text{ or } \exists j < i.\, \pi^j \models g & & &
\end{array}
$$

Since we are only concerned with finding witnesses for LTL formulae, we define an LTL formula f to hold at state $s \in \Sigma$ (written $s \models f$) iff there exists a path π in Σ^ω starting at $\pi(0) = s$ with $\pi \models f$. In addition we define f to hold in a set of states $S \subseteq \Sigma$ (written $S \models f$) iff there exists $s \in S$ with $s \models f$.

3 Single State Tableaux

In this section we present a variation on a tableau construction for explicit state model checking of LTL properties based on [2, 19]. The nodes in these *hybrid* tableaux also contain states of the model under investigation and not just temporal formulae. The main contribution of our paper is a symbolic extension to this tableau construction and is introduced Section 4.

The LTL model checking algorithm of [19] is the dual to the tableau construction of [2]. In [2] universal path quantifiers are considered, whereas [19] and our approach solve the dual model checking problem for existential path quantifiers. A tableau in the sense of [19] can be transformed into a tableau of [2] by replacing every literal by its negation, every temporal operator by its dual, e.g. every occurrence of **F** by **G**, every **E** by **A**, and every boolean operator by its dual as well, e.g. $\wedge$ by $\vee$. In essence this transformation is just a negation of every formula in the tableau. The terminology of a successful tableau has also to be revised, which can be found further down.

We call the type of tableau of [19] a *single state tableau* (S-Tableau), since every node in the tableau only contains a single state. Our tableau construction, introduced in section 4, allows a set of states at each tableau node. These tableaux are called *multiple state tableaux* (M-Tableau). In the rest of this section we will present a slight modification of the S-Tableau construction of [19] and note some facts that can be derived from [2, 19].

$$R_{\mathcal{A}^+}\colon \frac{s \vdash \mathbf{E}(\Phi, p)}{s \vdash \mathbf{E}(\Phi)} \quad p \in \ell(s), \quad p \in \mathcal{A} \qquad\qquad R_{\mathcal{A}^-}\colon \frac{s \vdash \mathbf{E}(\Phi, \neg p)}{s \vdash \mathbf{E}(\Phi)} \quad p \notin \ell(s), \quad p \in \mathcal{A}$$

$$R_{\mathbf{U}}\colon \frac{s \vdash \mathbf{E}(\Phi, f\,\mathbf{U}\,g)}{s \vdash \mathbf{E}(\Phi, g) \qquad s \vdash \mathbf{E}(\Phi, f, \mathbf{X} f\,\mathbf{U}\,g)} \qquad\qquad R_{\wedge}\colon \frac{s \vdash \mathbf{E}(\Phi, f \wedge g)}{s \vdash \mathbf{E}(\Phi, f, g)}$$

$$R_{\mathbf{R}}\colon \frac{s \vdash \mathbf{E}(\Phi, f\,\mathbf{R}\,g)}{s \vdash \mathbf{E}(\Phi, f, g) \qquad s \vdash \mathbf{E}(\Phi, g, \mathbf{X} f\,\mathbf{R}\,g)} \qquad\qquad R_{\vee}\colon \frac{s \vdash \mathbf{E}(\Phi, f \vee g)}{s \vdash \mathbf{E}(\Phi, f) \qquad s \vdash \mathbf{E}(\Phi, g)}$$

$$R_{\mathbf{F}}\colon \frac{s \vdash \mathbf{E}(\Phi, \mathbf{F} f)}{s \vdash \mathbf{E}(\Phi, f) \qquad s \vdash \mathbf{E}(\Phi, \mathbf{XF} f)} \qquad\qquad R_{\mathbf{G}}\colon \frac{s \vdash \mathbf{E}(\Phi, \mathbf{G} f)}{s \vdash \mathbf{E}(\Phi, f, \mathbf{XG} f)}$$

$$R_{\mathbf{X}}\colon \frac{s \vdash \mathbf{E}(\mathbf{X}\Phi_1, \ldots, \mathbf{X}\Phi_n)}{s_1 \vdash \mathbf{E}(\Phi_1, \ldots, \Phi_n) \quad \ldots \quad s_m \vdash \mathbf{E}(\Phi_1, \ldots, \Phi_n)} \quad \{s_1, \ldots, s_m\} = \mathrm{Img}(\{s\})$$

$$R_{\text{split}}\colon \frac{s \vdash \mathbf{E}(\Phi)}{s \vdash \mathbf{E}(\Phi)}$$

Fig. 1. S-Rules: Single state tableau rules.

A *single state sequent* (S-Sequent) consists of a single state s and a list of LTL formulae $\Phi = (\Phi_1, \ldots, \Phi_n)$, written $s \vdash \mathbf{E}(\Phi)$. The order of the formulae in the list is not important. An S-Sequent $s \vdash E(\Phi)$ holds in a Kripke structure K iff $s \models \Phi_1 \wedge \cdots \wedge \Phi_n$ in K. An S-Tableau is a finite directed graph of nodes labeled with S-Sequents that are connected via the rules shown in figure 1. The application of a rule results in edges between the premise and each conclusion. If not otherwise stated, we assume that a tableau is fully expanded, i.e. no rule can generate new sequents or edges. For technical reasons a sequent may occur several times in a tableau. In particular, we assume that two nodes labeled with the same sequent are connected with an edge generated by application of the split rule R_{split}. This assumption allows us to extract an S-Tableau from an M-Tableau without shortening paths in the tableau. Refer to Lemma 6 for more details. We further assume that the split rule is only applied finitely often which keeps the tableau finite, since the number of different sequents is finite and nodes labeled with identical sequents can only be introduced by the split rule.

An S-Path x is defined as the sequence of labels (sequents) of the nodes on a path through an S-Tableau. Since a path through a tableau can be both, finite or infinite, the same applies to an S-path. A finite S-Path x is *successful* iff it ends with a sequent $s \vdash \mathbf{E}()$, where the list of formulae is empty. An S-Tableau is *partially successful* iff the tableau contains a finite successful S-Path. It is *partially unsuccessful* iff no finite S-Path is successful.

An eventuality f, contained in the list of formulae of $x(i)$, is called *fulfilled* iff there exists j with $i \leq j < |x|$ and the body of f is contained in $x(j)$. This definition also applies to infinite paths. An infinite S-Path x is called *successful* iff every eventuality in x is fulfilled. Finally a tableau is *successful* iff it is partially successful or it contains an infinite successful S-Path. With these definitions we can derive the following theorem from [2, 19]:

Theorem 1. *An S-Tableau with root $s \vdash \mathbf{E}(f)$ is successful iff $s \models f$.*

This theorem implies that $s \models f$ iff *every* S-Tableau with root $s \vdash \mathbf{E}(f)$ is successful. Therefore completeness and correctness is independent of the order in which the rules are applied. In the construction of the tableau no backtracking is required. The freedom to choose a rule, if several are applicable, can be used to minimize the size of the tableau, and thus the running time of the model checking algorithm (see Section 6 and Section 7).

Finding successful paths seems to be an algorithmically complex problem. However, we will show that this problem can be reduced to the search for a successful strongly connected component. A strongly connected component (SCC) of a directed graph is a maximal subgraph in which every node can be reached from every other node in the subgraph. Note that in our notation a single node, not contained in any cycle, is not an SCC. We call an SCC of an S-Tableau successful iff every eventuality occurring in the SCC is fulfilled in the SCC, i.e. with every eventuality the SCC also contains its body. For an efficient implementation the following theorem is very important. As a result the search for a successful infinite path can be replaced by the search for a successful SCC, which is algorithmically much simpler.

Theorem 2. *A partially unsuccessful S-Tableau is successful iff it contains a successful SCC.*

This theorem is an easy consequence of the following Lemma, which can be used for the generation of an infinite witness, respectively counterexample, as explained below.

Lemma 3. *An S-Tableau contains an infinite successful S-Path iff it contains a successful SCC.*

Proof. For the proof from left to right let x be an infinite successful S-Path. First note that there has to be an SCC C in which a suffix of x is fully contained, since the S-Tableau is finite. Further let σ be a sequent in C with an eventuality f on the right hand side (RHS). We need to show that f is fulfilled in C. Since C is an SCC, there exists a path segment connecting σ to the start of the suffix of x in C. If the body of f occurs on this path segment then we are done. Otherwise, by the structure of the rules for eventualities, the first sequent of the suffix of x in C reached by this segment still contains f or $\mathbf{X}f$. Since x is successful the body of f has to occur in the suffix of x which is part of C.

The other direction is proven by constructing a successful S-Path from a successful SCC. This is easily done by generating a cyclic path segment through the SCC containing every sequent in the SCC. From the start of the cycle we can find another path segment back to the root of the tableau. Combining these two segments, repeating the cyclic segment infinitely often, results in an infinite successful S-path. □

If the tableau is unsuccessful then the root sequent can not hold. If the tableau is successful then it contains a finite successful path or a successful SCC. In the first case we can extract a finite witness for the root sequent by extracting a list of states from the finite successful path. In the second case we extract the witness from the S-Path generated in the second part of the proof for Lemma 3.

If we apply our approach to a universal model checking problem by using the negation of the universal property to be checked in the root of the tableau, then the procedure described in the previous paragraph serves as an algorithm for generating counterexamples.

To find the successful SCCs of an S-Tableau in linear time, a variation of the standard algorithm of Tarjan for the decomposition of a directed graph into its strongly connected components can be used. In particular, whenever a new SCC in Tarjan's algorithm is found, we check on-the-fly if it is successful. Thus the model checking problem can be solved in linear time in the size of the tableau as well. The size of the tableau is bounded by the number of different S-Sequents. Note that the split rule is never applied in this context. The number of different S-Sequents in a tableau with root $s \vdash \mathbf{E}(f)$ is in $\mathbf{O}(|\Sigma| \cdot 2^{|f|})$, since the RHS of a sequent may contain an arbitrary subset of subformulae of f. This gives an explicit state model checking algorithm with worst case complexity linear in the size of the Kripke structure and exponential in the size of the formula. For more details compare with [2].

A *cyclic path* is an infinite path of the form $A \cdot B^{\omega}$, that starts with a finite prefix A and continues with a path segment B repeated infinitely often. Not all infinite paths through a tableau are *cyclic paths*. However, the proof of Lemma 3 shows that it is enough to consider cyclic paths only, when looking for successful paths to determine whether a tableau is successful.

4 Multiple State Tableaux

In this section we extend the tableau construction of the last section to handle multiple states in one sequent. In combination with a symbolic representation, such as BDDs, this extension potentially leads to an exponential reduction in tableau size. The idea of handling set of states on the left hand side (LHS) of sequents already occurred in [4] as an extension of local model checking for the modal μ-calculus [28]. The tableau construction in this section extends the LTL model checking algorithm of [2] in a similar way.

$$R_{\mathcal{A}^+}\colon \frac{S \vdash \mathbf{E}(\Phi, p)}{S_p^+ \vdash \mathbf{E}(\Phi)} \qquad\qquad R_{\mathcal{A}^-}\colon \frac{S \vdash \mathbf{E}(\Phi, \neg p)}{S_p^- \vdash \mathbf{E}(\Phi)}$$

$$R_{\mathbf{U}}\colon \frac{S \vdash \mathbf{E}(\Phi, f\,\mathbf{U}\,g)}{S \vdash \mathbf{E}(\Phi, g) \qquad S \vdash \mathbf{E}(\Phi, f, \mathbf{X}f\,\mathbf{U}\,g)} \qquad\qquad R_{\wedge}\colon \frac{S \vdash \mathbf{E}(\Phi, f \wedge g)}{S \vdash \mathbf{E}(\Phi, f, g)}$$

$$R_{\mathbf{R}}\colon \frac{S \vdash \mathbf{E}(\Phi, f\,\mathbf{R}\,g)}{S \vdash \mathbf{E}(\Phi, f, g) \qquad S \vdash \mathbf{E}(\Phi, g, \mathbf{X}f\,\mathbf{R}\,g)} \qquad\qquad R_{\vee}\colon \frac{S \vdash \mathbf{E}(\Phi, f \vee g)}{S \vdash \mathbf{E}(\Phi, f) \qquad S \vdash \mathbf{E}(\Phi, g)}$$

$$R_{\mathbf{F}}\colon \frac{S \vdash \mathbf{E}(\Phi, \mathbf{F}f)}{S \vdash \mathbf{E}(\Phi, f) \qquad S \vdash \mathbf{E}(\Phi, \mathbf{XF}f)} \qquad\qquad R_{\mathbf{G}}\colon \frac{S \vdash \mathbf{E}(\Phi, \mathbf{G}f)}{S \vdash \mathbf{E}(\Phi, f, \mathbf{XG}f)}$$

$$R_{\text{split}}\colon \frac{S \vdash \mathbf{E}(\Phi)}{S_1 \vdash \mathbf{E}(\Phi) \quad \cdots \quad S_k \vdash \mathbf{E}(\Phi)} \quad S = S_1 \,\dot\cup \cdots \dot\cup\, S_k, \quad S_i \neq \{\}, \text{ for } i = 1 \ldots k$$

$$R_{\mathbf{X}}\colon \frac{S \vdash \mathbf{E}(\mathbf{X}\Phi_1, \ldots, \mathbf{X}\Phi_n)}{T \vdash \mathbf{E}(\Phi_1, \ldots, \Phi_n)} \quad T = \mathrm{Img}(S)$$

Fig. 2. M-Rules: Multiple state tableau rules ('S_p^+' and 'S_p^-' are defined in the text).

A *multiple state sequent* (M-Sequent) consists of a set of states S and a list of LTL formulae $\Phi = (\Phi_1, \ldots, \Phi_n)$, written $S \vdash \mathbf{E}(\Phi)$. We use the same symbol '$\vdash$' to separate left and right hand side of S-Sequents and M-Sequents, but capital letters, e.g. S, for set of states on the LHS of M-Sequents and lower case letters, e.g. s, for S-Sequents. An M-Sequent *holds* in a Kripke structure K iff $S \models \Phi_1 \wedge \cdots \wedge \Phi_n$, i.e. there exists a path π in K with $\pi(0) \in S$ and $\pi \models \Phi_1 \wedge \cdots \wedge \Phi_n$. An M-Tableau is a rooted finite directed graph of nodes labeled with M-Sequents that are connected via the rules shown in figure 2, where we define the following short hand for $p \in \mathcal{A}$:

$$S_p^+ := \{s \in S \mid p \in \ell(s)\}, \qquad S_p^- := \{s \in S \mid p \notin \ell(s)\}$$

In the split rule R_{split} the set of states S on the LHS is partitioned into a nonempty pairwise disjunctive list of sets $S_1, \ldots, S_k$ that cover S. For M-Tableaux we require every node to be labeled with a unique M-Sequent. M-Paths, successful M-Path, and SCC are defined exactly as in the single state case of the last section. The only exception is a finite M-Path ending with an M-Sequent $\{\} \vdash E(\Phi)$, with an empty set of states on the left side. By definition, such an M-Path is always unsuccessful even if the list of formulae Φ is empty.

To lift Theorem 1, Theorem 2, and in particular Lemma 3 to M-Tableaux we first note that the definitions of successful paths, SCCs, and successful tableau do only depend on the RHS of the sequents, in both cases, for S-Tableaux and M-Tableaux. As an immediate consequence we have:

Theorem 4. *A partially unsuccessful M-Tableau is successful iff it contains a successful SCC.*

Lemma 5. *An M-Tableau contains an infinite successful M-Path iff it contains a successful SCC.*

The second step is to map an M-Tableau to a *set* of S-Tableau, where the M-Tableau is successful iff one S-Tableau is successful. The mapping Ψ_0 is defined along the graph structure of the M-Tableau. If $\sigma = (\{r_1, \ldots, r_n\} \vdash \mathbf{E}(\Phi))$ is the root sequent of the M-Tableau, the result of the mapping will be n S-Tableaux with roots $\sigma_i = (r_i \vdash \mathbf{E}(\Phi))$ for $i = 1 \ldots n$. Now we apply the rule that has been applied to the root M-sequent to each individual root S-sequent as well, obtaining valid successor sequents in the S-Tableau. Then the newly generated nodes are extended by applying the same rule as in the M-Tableau. This process is repeated until the constructed tableau can not be extended anymore.

Let $\mathcal{T}$ be an M-Tableau with root σ, as above, then $\Psi_0(\mathcal{T})$ is defined as the set of S-Tableaux

$$\{\Psi(\mathcal{T}, \sigma, \sigma_1), \ldots, \Psi(\mathcal{T}, \sigma, \sigma_n)\}$$

where Ψ is defined along the graph structure of $\mathcal{T}$ starting with the root sequent σ, as defined below. Note that Ψ returns a single S-Tableau while Ψ_0 returns a set of S-Tableaux.

To define Ψ we map every instance of an M-Rule in the M-Tableau to an application of the corresponding S-Rule in the S-Tableau using the fact that in an M-Tableaux every node can be identified uniquely by its label. Let $\sigma_M = (S \vdash \mathbf{E}(\Phi_M))$ be an M-Sequent of $\mathcal{T}$ and $\sigma_S = (s \vdash \mathbf{E}(\Phi_S))$ be an S-Sequent. Then $\Psi(\mathcal{T}, \sigma_M, \sigma_S)$ is only defined iff $s \in S$ and $\Phi = \Phi_S = \Phi_M$. Now let R_Λ be the M-Rule that is applied in $\mathcal{T}$ to σ_M:

$$R_\Lambda\colon \frac{S \vdash \mathbf{E}(\Phi)}{S_1 \vdash \mathbf{E}(\Phi_1) \quad \cdots \quad S_k \vdash \mathbf{E}(\Phi_k)}$$

By definition of the rules, if $\Lambda \in \{\mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{G}, \wedge, \vee\}$, then R_Λ is applicable to σ_S as an S-Rule and yields:

$$R_\Lambda\colon \frac{s \vdash \mathbf{E}(\Phi)}{s \vdash \mathbf{E}(\Phi_1) \quad \cdots \quad s \vdash \mathbf{E}(\Phi_k)}$$

In this case $s \in S = S_1 = \cdots = S_k$ and we continue our construction by expanding the S-Sequent $s \vdash \mathbf{E}(\Phi_i)$ by $\Psi(\mathcal{T}, S \vdash \mathbf{E}(\Phi_i), s \vdash \mathbf{E}(\Phi_i))$ for $i = 1 \ldots k$. If $\Lambda = \text{split}$, then there exists an j with $s \in S_j$, since the partition covers S. Therefore we can apply the S-Rule R_{split} on σ_S which yields a new node labeled with σ_S again. This new node is expanded by $\Psi(\mathcal{T}, S_j \vdash \mathbf{E}(\Phi), \sigma_S)$.

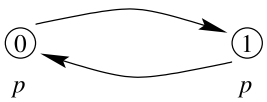
Regarding the rules for atomic propositions we only consider the positive case $R_{\mathcal{A}^+}$. The definition of Ψ for $R_{\mathcal{A}^-}$ is similar. If $R_{\mathcal{A}^+}$ is applicable to σ_S then we proceed as above. Otherwise the construction of the S-Tableau is terminated with an unsuccessful finite path and $\Psi(\mathcal{T}, \sigma_M, \sigma_S)$ consists of a single node labeled with σ_S. Finally consider the $R_\mathbf{X}$ rule that involves the image operator. Let $T = \text{Img}(S)$, $T_s = \text{Img}(\{s\}) = \{t_1, \ldots, t_m\} \subseteq T$, $A = (\Phi_1 \ldots, \Phi_l)$, and $\mathbf{X}A = (\mathbf{X}\Phi_1, \ldots, \mathbf{X}\Phi_l)$.

$$R_{\mathbf{X}}\colon \frac{S \vdash \mathbf{E}(\mathbf{X}A)}{T \vdash \mathbf{E}(A)} \qquad\qquad R_{\mathbf{X}}\colon \frac{s \vdash \mathbf{E}(\mathbf{X}A)}{t_1 \vdash \mathbf{E}(A) \quad \cdots \quad t_m \vdash \mathbf{E}(A)}$$

And again we expand the nodes labeled $t_i \vdash \mathbf{E}(A)$ with $\Psi(\mathcal{T}, T \vdash \mathbf{E}(A), t_i \vdash \mathbf{E}(A))$. This simple recursive definition of Ψ may result in an infinite S-Tableau. We can keep the result finite if we exit the recursion by introducing a loop as soon as the same arguments to Ψ occur the second time, as in the following example for mapping an M-Tableau into its corresponding S-Tableaux.

Consider the Kripke structure K with two states 0 and 1, both initial states, and two transitions from state 0 to state 1 and from state 1 to state 0. Both states are labeled with p, the only atomic proposition. An M-Tableau for checking $\mathbf{EG}p$ looks as follows

$$\begin{array}{l} \{0,1\} \vdash \mathbf{E}(\mathbf{G}p) \\ \hline \{0,1\} \vdash \mathbf{E}(p, \mathbf{XG}p) \\ \hline \{0,1\} \vdash \mathbf{E}(\mathbf{XG}p) \end{array}$$



and the application of $\mathbf{R_X}$ to the leaf sequent leads back to the root sequent. The tableau represents one successful M-Path that contains only one image calculation. The given M-Tableau is mapped into the following two S-Tableaux:

$$\begin{array}{l} 0 \vdash \mathbf{E}(\mathbf{G}p) \\ \hline 0 \vdash \mathbf{E}(p, \mathbf{XG}p) \\ \hline 0 \vdash \mathbf{E}(\mathbf{XG}p) \\ \hline 1 \vdash \mathbf{E}(\mathbf{G}p) \\ \hline 1 \vdash \mathbf{E}(p, \mathbf{XG}p) \\ \hline 1 \vdash \mathbf{E}(\mathbf{XG}p) \end{array} \qquad\qquad \begin{array}{l} 1 \vdash \mathbf{E}(\mathbf{G}p) \\ \hline 1 \vdash \mathbf{E}(p, \mathbf{XG}p) \\ \hline 1 \vdash \mathbf{E}(\mathbf{XG}p) \\ \hline 0 \vdash \mathbf{E}(\mathbf{G}p) \\ \hline 0 \vdash \mathbf{E}(p, \mathbf{XG}p) \\ \hline 0 \vdash \mathbf{E}(\mathbf{XG}p) \end{array}$$

Again the application of $\mathbf{R_X}$ to the leaf nodes yields the root. In general, mapping an M-Tableau may produce larger tableaux.

For each generated S-Tableau it is easy to construct a graph homomorphism λ that maps nodes and edges of the S-Tableau to the nodes, respectively edges, of the M-Tableau, with the following property: If $\lambda(n_S) = n_M$, where n_S is a node of the S-Tableau, labeled with the S-Sequent $s \vdash \mathbf{E}(\Phi_S)$, and n_M is a node in the M-Tableau, labeled with an M-Sequent $S \vdash \mathbf{E}(\Phi_M)$, then $s \in S$ and $\Phi_M = \Phi_S$. Further every edge, that was generated by the application of an S-Rule R_Λ is mapped into an edge of the M-Tableau that was generated by the M-Rule R_Λ.

Let $\mathcal{T}_M$ be an M-Tableau and $\mathcal{T}_S$ be an S-Tableau with $\mathcal{T}_S \in \Psi_0(\mathcal{T}_M)$. Then we say that $\mathcal{T}_S$ *matches* $\mathcal{T}_M$. We call an S-Path x a *matching path* to an M-Path X iff the S-Tableau in which x occurs matches the M-Tableau of X and $\lambda(x) = X$ for the corresponding graph homomorphism λ.

Lemma 6. *Let X be a successful finite or cyclic M-Path in an M-Tableau $\mathcal{T}_M$. Then there exists an S-Tableau $\mathcal{T}_S$ that matches $\mathcal{T}_M$ and contains an successful S-Path x matching X.*

Proof. First let X be a finite successful path. Then X ends with a sequent $S \vdash \mathbf{E}()$ with $S \neq \{\}$. We pick an arbitrary state $s \in S$ and traverse X backward until the root is reached. At $X(i)$ we generate the S-Sequent $x(i)$ starting with $s \vdash \mathbf{E}()$.

Let $X(i) = (S_i \vdash \mathbf{E}(\Phi_i))$, then we generate $x(i) = (s_i \vdash \mathbf{E}(\Phi_i))$ with $s_i \in S_i$ and the following restriction. As long as no $R_{\mathbf{X}}$ rule is applied to $X(i)$ we define the LHS of $x(i)$ to be the state on the LHS of $x(i+1)$. If $R_{\mathbf{X}}$ is applied to $X(i)$ then we define s_i as an arbitrary predecessor of s_{i+1} in S_i.

Second let $X = Y \cdot Z^\omega$ be a cyclic successful path. The generated x will also be of the form $x = y \cdot z^\omega$, where y matches Y and z matches Z^n for some $n > 0$. We start with an arbitrary $s \in Z(m)$ where $m = |Z| - 1$ and traverse Z backward until $Z(0)$ is reached, generating S-Sequents as in the finite case. After we have reached $Z(0)$ we continue at $Z(m)$. This process is repeated until the same S-Sequent was generated twice during the visit of $Z(0)$. Termination is guaranteed because the set of states in $Z(0)$ is finite. Then z is defined as the path segment from the last occurrence of the same S-Sequent to the first. From $z(0)$ we can find a finite prefix y to the root as in the finite case.

By construction $\mathcal{T}_S = \Psi(\mathcal{T}_M, X(0), x(0))$ is defined, matches $\mathcal{T}_M$, contains the successful S-Path x, and x matches X. $\square$

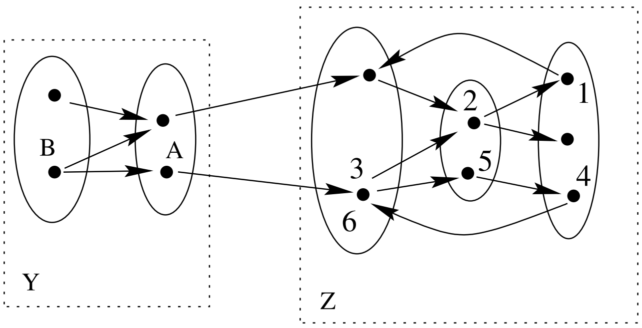


Fig. 3. Extracting an S-Path from an M-Path.

Consider the example of figure 3 where each ellipsis depicts the LHS of a sequent on which the $R_{\mathbf{X}}$ rule is applied. The small filled circles represent single states of the Kripke structure. The arrows between those circles are transitions of the Kripke structure. Thus the picture visualizes a sequence $Y \cdot Z^{\omega}$ of set of states with $Y(1) = \text{Img}(Y(0))$ and

$$Z(0) = \text{Img}(Y(1)), \quad Z(1) = \text{Img}(Z(0)), \quad Z(2) = \text{Img}(Z(1)), \quad Z(0) = \text{Img}(Z(2))$$

Our goal is to extract a sequence of single states from $Y \cdot Z^{\omega}$. We start with 1, transition to 2 and pick 3 as predecessor of 2. The next transition, from 3 to 4, brings us back to the last sequent of Z but no cycle can be closed yet. We continue with 5 and reach 3 again with 6. From there we find a prefix (B, A), that leads from the initial state B to the start of the cycle at 6. The resulting witness is $(\text{B}, \text{A}) \cdot (6, 5, 4)^{\omega}$.

Lemma 6 allows us to derive the following completeness and correctness result for M-Tableau. Recall that $S \models f$ iff there exists $s \in S$ with $s \models f$.

Theorem 7. *An M-Tableau with root $S \vdash \mathbf{E}(f)$ is successful iff $S \models f$.*

Proof. If an M-Tableau $\mathcal{T}_M$ is successful then it contains a successful path. Using Lemma 6 we can construct a matching successful path in a matching S-Tableau $\mathcal{T}_S$ with root sequent $s \vdash \mathbf{E}(f)$ and $s \in S$. The correctness of the S-Tableaux construction (one part of Theorem 1) proves $s \models f$ which in turn implies $S \models f$.

Now let $\mathcal{T}_M$ be unsuccessful. Then for every $s \in S$ every matching S-Tableau $\mathcal{T}_S$ with root $s \vdash \mathbf{E}(f)$ is unsuccessful as well. With Theorem 1 we conclude $s \not\models f$ for all $s \in S$. $\square$

The algorithm described in the proof of Lemma 6 can be used to construct a witness for the root sequent from a successful M-Tableau, or a counterexample for the negation of the root sequent. First a matching path is constructed. Then a witness, a finite or cyclic path of states, can easily be extracted.

5 Algorithm

A more detailed description of the tableau construction follows in this section. The overall approach expands open branches in DFS manner and stops when a successful finite path has been generated, a successful SCC has been found, or finally no rule can be applied without regenerating an edge that already exists in the tableau. In the first two cases a witness for the root sequent can be generated. In the last case it is shown that the root sequent can not hold.

Finite successful paths are easy to detect. To detect and not to miss any successful SCC we use a modified version of Tarjan's algorithm for decomposing a directed graph into its strongly connected components. It is the same algorithm as used for S-Tableau in Section 3.

During the construction we have to remember the sequents that already occurred in the tableau. This can be accomplished by a partial function mapping a sequent to a node. To implement this we can sort the sequents in the tableau, use a hash table, or simply an array. In practice a hash table is the best solution.

Up to this point the algorithm is identical for both, single and multiple state tableaux. Our intention, of course, is to represent set of states with BDDs. We associate with each formula $\mathbf{E}(\Phi)$ the list of sequents in the tableau that have $\mathbf{E}(\Phi)$ on the RHS. To check if a sequent $\sigma \equiv (S \vdash \mathbf{E}(\Phi)$ already occurred, we just go through the list of sequents associated with $\mathbf{E}(\Phi)$ and check whether the BDD representing the set of states on the LHS of one of the sequents in the list is the same as the BDD representing S.

6 Heuristics

The rule R_{split} is not really necessary for the completeness but it helps to reduce the search space, i.e. the size of the generated tableau. For instance consider the construction of a tableau for the formula $\mathbf{EF}p$. This formula is the negation of a simple safety property. In this case a good heuristics is to build the tableau by expanding the left successor of the rule $R_{\mathbf{F}}$ first. Only if the left branch does not yield a successful path, then the right successor is tried. If during this process a sequent $\sigma' = S' \vdash \mathbf{E}(\mathbf{F}f)$ is found and a sequent $\sigma'' = S'' \vdash \mathbf{E}(\mathbf{F}f)$ occurs on the path from the root to σ' and $S'' \subseteq S'$ then we can remove the set S'' from S' by applying R_{split} with $S_1 = S''$ and $S_2 = S' - S''$. The left successor immediately leads to an unsuccessful infinite path and we can continue with the right successor.

A successful path in a tableau for $\mathbf{EF}p$ is a counterexample for the dual safety property $\mathbf{AG}\neg p$. Thus applying the heuristic of the last paragraph essentially implements an algorithm that computes the set of reachable states in a BFS manner while checking on-the-fly for states violating the safety property (as the *early evaluation* technique in [1]). An example of this technique is shown in figure 4 using the Kripke structure of figure 5.

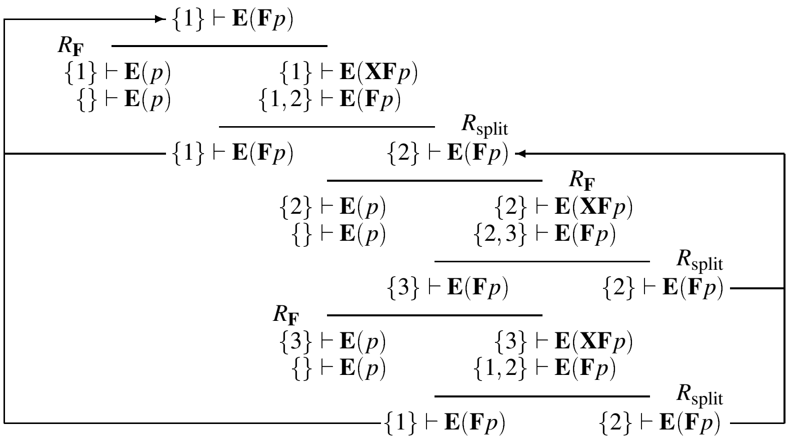


Fig. 4. Example tableau for on-the-fly model checking of safety properties.

Similar heuristics can be used for liveness properties. Consider the Kripke structure in figure 6 and the property $\mathbf{EG}p$. If we split a sequent as soon it LHS contains a state that already occurred in a sequent with

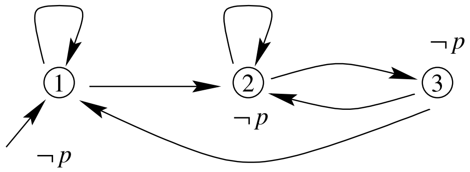


Fig. 5. Example Kripke structure for on-the-fly checking of safety properties.

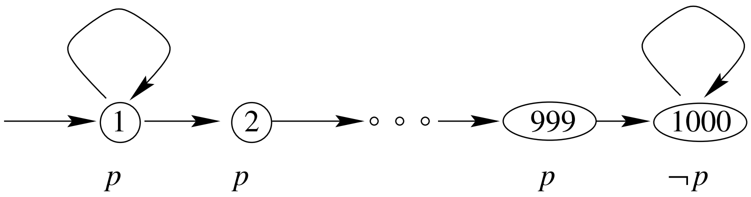


Fig. 6. Example Kripke structure for on-the-fly model checking of liveness properties.

the same RHS, then the following tableau finds a witness for $\mathbf{EG}p$ with just one image computation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\{1\} \vdash \mathbf{E}(\mathbf{G}p)}{\{1\} \vdash \mathbf{E}(p, \mathbf{XG}p)}\,R_{\mathbf{G}}}{\{1\} \vdash \mathbf{E}(\mathbf{XG}p)}\,R_{\mathcal{A}^+}}{\{1,2\} \vdash \mathbf{E}(\mathbf{G}p)}\,R_{\mathbf{X}}}{\{1\} \vdash \mathbf{E}(\mathbf{G}p) \qquad \{2\} \vdash \mathbf{E}(\mathbf{G}p)}\,R_{\text{split}}$$

The left sequent that results from the application of the split rule is the same as the root sequent. Thus a successful infinite path has been found and the right branch does not need to be expanded. A model checking algorithm based on standard fixpoint calculations requires 1000 image computations, i.e. traverses the whole state space, before the witness can be found.

Another heuristic is to avoid splitting the tableau as long as possible. In particular, if several rules are applicable to a sequent, then, if possible, a rule is chosen that does not result in multiple branches. For example if both $R_{\mathbf{G}}$ and $R_{\mathbf{F}}$ are applicable, then $R_{\mathbf{G}}$ is always preferred. This is one of the heuristics proposed in [27] for the construction of small tableau as an intermediate step of translating LTL into the modal μ-calculus with the algorithm of [10]. In general, these heuristics are also applicable to our approach.

7 Optimization

In an M-Tableau the number of different LHS of sequents is exponential in $|\Sigma|$, the number of states of the Kripke structure. In this section we present an optimization that reduces the maximal number of different LHS with the same RHS to $2 \cdot |\Sigma|$. In particular, with this optimization the size of an M-Tableau becomes linear in the number of states. Without this optimization the tableau construction would not be feasible at all, even in combination with BDDs.

Note that the size of the tableau may still be exponential in the size of the formula, since the RHS of a sequent is an arbitrary set of subformulae of the original property. In practice the size of the properties to be checked is usually small, but the size of the models we deal with can be arbitrary large. Thus it is much more important to have an algorithm that is linear in the size of the model.

The basic idea is to maintain the following invariant by applying the split rule: Let $S_1 \vdash \mathbf{E}(\Phi)$ and $S_2 \vdash \mathbf{E}(\Phi)$ be two M-Sequents in the M-Tableau. Then $S_1 \cap S_2 = \{\}$, $S_1 \subseteq S_2$, or $S_2 \subseteq S_1$.

Lemma 8. *Let $P \subseteq \mathbb{P}(S)$ be a set of nonempty subsets of a finite set $S \neq \{\}$. If for all $S_1, S_2 \in P$ either $S_1 \cap S_2 = \{\}$, $S_1 \subseteq S_2$ or $S_2 \subseteq S_1$ holds, then $|P| < 2 \cdot |S|$.*

Proof (Induction over $|S|$). In the base case let $|P| \leq 1$, which implies $|P| < 2 \leq 2 \cdot |S|$. In the induction step, we assume $|P| > 1$. The sets contained in P are partially ordered by set inclusion and contain at least one maximal set S_1. By defining $S_2 := S \backslash S_1$ we partition $S = S_1 \cup S_2$ into two mutually exclusive sets, S_1 and S_2. If $S_2 = \{\}$, i.e. S_1 is the *only* maximal set in P, then we define

$$P' := P \backslash \{S_1\} \quad \text{and} \quad S' := \bigcup_{T \in P'} T$$

and with $0 < |S'| < |S_1| \leq |S|$ the induction hypothesis shows

$$|P| = 1 + |P'| < 1 + 2 \cdot |S'| < 2 \cdot (|S'| + 1) \leq 2 \cdot |S_1| \leq 2 \cdot |S|$$

Now we assume $S_2 \neq \{\}$, which implies $S_1 \neq S$, and induces a partition on P with

$$P = P_1 \cup P_2 \qquad \text{and} \qquad P_i := \{T \in P \mid T \subseteq S_i\} \quad \text{for } i = 1, 2$$

Finally the induction hypothesis applied to S_1 and S_2 results in

$$|P| \leq |P_1| + |P_2| < 2 \cdot |S_1| + 2 \cdot |S_2| = 2 \cdot (|S_1| + |S_2|) = 2 \cdot |S|$$

□

Since the LHS of a sequent can also be an empty set, Lemma 8 shows that the number of sequents with the same RHS is bounded by $2 \cdot |S|$ if the invariant is maintained. This is also the best bound that we can get, as the following example shows. Let $S_i = \{0, \ldots, 2^i - 1\}$. Then we associate a balanced binary tree of height i with each $P_i \subseteq \mathbb{P}(S_i)$. We label each node in the tree by exactly one element of P_i, which implies $|P_i| = 2^{i+1} - 1 = 2 \cdot |S_i| - 1$. The i leaves are labeled with the singleton sets $\{0\}, \ldots, \{2^i - 1\}$. Each inner node of the binary tree is labeled with the union of the labels of its children.

Note that the image rule $R_{\mathbf{X}}$ and the atomic rules $R_{\mathcal{A}^+}$ and $R_{\mathcal{A}^-}$ are the only rules, except the split rule R_{split}, that actually manipulate the LHS. All other rules only generate sequents that contain the same set of states as their parents. The application of these rules can not violate the invariant. To maintain the invariant in the case of $R_{\mathbf{X}}$, $R_{\mathcal{A}^+}$ and $R_{\mathcal{A}^-}$ as well we have to apply the split rule in combination with these rules: After each image calculation or application of an atomic rule the sequent is split to fulfill the invariant. For instance if the application of $R_{\mathbf{X}}$ yields:

$$R_{\mathbf{X}}\colon \frac{\cdots}{\{1,2,3,4\} \vdash \mathbf{E}(\Phi)}$$

and the tableau already contains the two sequents:

$$\{1,2\} \vdash \mathbf{E}(\Phi) \qquad \text{and} \qquad \{4,5\} \vdash \mathbf{E}(\Phi)$$

Then the split rule is applied as follows. For every non empty intersection of $\{1,2,3,4\}$ with the LHS of an already existing sequent a new sequent is generated:

$$R_{\text{split}}\colon \frac{\{1,2,3,4\} \vdash \mathbf{E}(\Phi)}{\{1,2\} \vdash \mathbf{E}(\Phi) \qquad \{3\} \vdash \mathbf{E}(\Phi) \qquad \{4\} \vdash \mathbf{E}(\Phi)}$$

where the first sequent is not actually generated since it already exists in the tableau.

We have to combine the R_{split} rule with the image and atomic rules to technically avoid introducing an intermediate sequent, $\{1,2,3,4\} \vdash \mathbf{E}(\Phi)$ in the example, that might violate the invariant. The correctness and completeness results can be proven for this modification as well. Without combining these rules each application of $R_{\mathbf{X}}, R_{\mathcal{A}^+}$ or $R_{\mathcal{A}^-}$ could potentially need an additional application of R_{split}. This could potentially double the number of sequents and the number of sequents with the same RHS can only be bounded by $4 \cdot |\Sigma|$, which is still linear in the number of states.

8 Complexity and Related Work

In this section we discuss the complexity of our new algorithm based on M-Tableaux and compare it with other local and global techniques for LTL model checking.

The size of a tableau with root $\Sigma_0 \vdash \mathbf{E}(f)$, not using the optimization of the last section, is in $\mathbf{O}(2^{|\Sigma|} \cdot 2^{|f|})$. The time taken is polynomial in the size of the tableau. Thus the time complexity is (roughly) the same as the space complexity. With the optimization of the last section the size of the tableau is reduced to $\mathbf{O}(|\Sigma| \cdot 2^{|f|})$. As a consequence the time complexity of our algorithm is at most polynomial in the number of states, with a small degree polynomial, and exponential in the size of the formula. The explicit state model checking algorithms of [2, 16, 21] are linear in the number of states and the number of transitions. Note that the number of transitions may be quadratic in the number of states. If an explicit state representation is used, we conjecture that our tableau construction can be implemented with the same linear time complexity. However, with our approach we are able to use compact data structures, such as BDDs, to represent sets of states symbolically and thus can hope to achieve exponentially smaller tableaux and exponentially smaller running times for certain examples.

The method of [10] translates an LTL formula into a tableau similar to the tableaux in our approach. In [10] the nodes contain only formulae and no states. The size of the tableau can be exponential in the size of the LTL formula. The second step is a translation of the generated tableau into a μ-calculus formula that is again exponential in the size of the tableau. Additionally, the alternation depth of the μ-calculus formula can not be restricted. With [13, 22] this results in a model checking algorithm with time and space complexity that is double exponential in the size of the formula and single exponential in the size of the model K.

In [15] an ELTL formula is translated to a Büchi automata with the method of [29]. This leads to an exponential blow up in the worst case. But see [14] for an argument why this explosion might not happen in practice, which also applies to our approach. The resulting Büchi automata is translated to *post*-μ, a forward version of the standard modal μ-calculus, for which similar complexity results for model checking as in [13, 22] can be derived. This translation produces a μ-calculus formula of alternation depth 2 which results in an algorithm with an at least quadratic running time in $|\Sigma|$.

The LTL model checking algorithm of [15] is also forward oriented. A forward state space traversal potentially avoids searching through non reachable states, as it is usually the case with simple backward approaches. However, it is not clear how DFS can be incorporated into symbolic μ-calculus model checking.

The method of [7] translates an LTL model checking problem into a FairCTL model checking problem. With the result of [13] this leads to a model checking algorithm that is linear in the size of the model and exponential in the size of the formula. Again, these complexity results are only valid for explicit state model checking. The algorithms of [7, 15] are based on BFS and it is not clear how to implement them depth first.

The work by Iwashita [17, 18] does not handle full LTL and no complexity analysis is given. But if we restrict our algorithm to the path expressions of [17, 18], then our algorithm subsumes the algorithms of [17, 18], even for the *layered approach* of [18].

In [4] an M-Tableau construction for the modal μ-calculus was presented. The main motivation in [4] for using set of states in sequents was to be able to handle infinite state systems. Therefore no complexity results were given. In addition, the modal μ-calculus, as already discussed above, can not represent LTL properties directly without a prior translation [10, 29].

Our tableau construction is on-the-fly (see liveness example in Section 6) and only needs $\mathbf{O}(|\Sigma|)$ image computations. Previous symbolic model checking algorithms for LTL [7, 15], based on fixpoint calculations, require $\mathbf{O}(|\Sigma|^2)$ image computations.

9 Conclusion

Although our technique clearly extends the work of [17, 18] and bridges the gap between local and global model checking, we still need to show that it works in practice. We are currently working on proving the conjecture that our tableau construction can be implemented with linear complexity. We also want to investigate heuristics for applying the split rule. The approximation techniques of [25, 26] are a good starting point.

References

[1] I. Beer, S. Ben-David, C. Eisner, D. Geist, L. Gluhovsky, T. Heyman, A. Landver, P. Paanah, Y. Rodeh, G. Ronin, and Y. Wolfsthal. Rulebase: Model checking at IBM. In *CVA'97*, number 1254 in LNCS, pages 480–483. Springer-Verlag, 1997.

[2] G. Bhat, R. Cleaveland, and O. Grumberg. Efficient on-the-fly model checking for CTL*. In *LICS'95*. IEEE Computer Society, 1995.

[3] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS'99*, LNCS. Springer, 1999.

[4] J. Bradfield and C. Stirling. Local model checking for infinite state spaces. *Theorectical Computer Science*, 96:157–174, 1992.

[5] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8), 1986.

[6] J. R. Burch, E. M. Clarke, and K. L. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98, 1992.

[7] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. *Formal Methods in System Design*, 10:47–71, 1997.

[8] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs: Workshop*, LNCS. Springer, 1981.

[9] R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27, 1990.

[10] M. Dam. CTL* and ECTL* as fragments of the modal mu-calculus. *Theoretical Computer Science*, 126, 1994.

[11] D. L. Dill. The MurΦ verification system. In *CAV'96*, LNCS. Springer, 1996.

[12] Y. Dong, X. Du, Y.S. Ramakrishna, C. T. Ramkrishnan, I. V. Ramakrishnan, S. A. Smolka, O. Sokolsky, E. W. Starck, and D. S. Warren. Fighting livelock in the i-protocol: A comparative study of verification tools. In *TACAS'99*, LNCS. Springer, 1999.

[13] E. A. Emerson and C.-L. Lei. Modalities for model checking: Branching time strikes back. *Science of Computer Programming*, 8, 1986.

[14] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings 15th Workshop on Protocol Specification, Testing, and Verification*. North-Holland, 1995.

[15] T. A. Henzinger, O. Kupferman, and S. Qadeer. From Pre-historic to Post-modern symbolic model checking. In *CAV'98*, LNCS. Springer, 1998.

[16] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 5(23), 1997.

[17] H. Iwashita and T. Nakata. CTL model checking based on forward state traversal. In *ICCAD'96*. ACM, 1996.

[18] H. Iwashita, T. Nakata, and F. Hirose. Forward model checking techniques oriented to buggy design. In *ICCAD'97*. ACM, 1997.

[19] A. Kick. *Generierung von Gegenbeispielen und Zeugen bei der Modellprüfung*. PhD thesis, Fakultät für Informatik, Universität Karlsruhe, 1996.

[20] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27, 1983.

[21] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Symposium on Principles of Programming Languages*, New York, 1985. ACM.

[22] D. E. Long, A. Browne, E. M. Clarke, S. Jha, and W. R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In *CAV'94*, LNCS. Springer, 1994.

[23] K. L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.

[24] J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. 5th Int. Symp. in Programming*, 1981.

[25] K. Ravi, K. L. McMillan, T. R. Shiple, and F. Somenzi. Approximation and decomposition of binary decision diagrams. In *DAC'98*. ACM, 1998.

[26] K. Ravi and F. Somenzi. High-density reachability analysis. In *ICCAD'95*. ACM, 1995.

[27] F. Reffel. Modellprüfung von Unterlogiken von CTL*. Masterthesis, Fakultät für Informatik, Universität Karlsruhe, 1996.

[28] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89, 1991.

[29] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1), 1994.