URL: http://www.elsevier.nl/locate/entcs/volume23.html 12 pages

Combining Local and Global Model Checking

Armin Biere, [a,b,c] Edmund M. Clarke, [b,c] Yunshan Zhu [b,c]

[a] *Institut für Logik, Komplexität und Deduktionssysteme (ILKD), University of Karlsruhe, Postfach 6980, 76128 Karlsruhe, Germany*

[b] *Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A*

[c] *Verysys Design Automation, Inc, 42707 Lawrence Place, Fremont, CA 94538*

Abstract

The verification process of reactive systems in *local model checking* [1,7] and in *explicit state model checking* [13,15] 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* [6,23] and, in particular, *symbolic model checking* [4,22] can utilize compact representations of the state space, e.g. BDDs [3], 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 [1] that enables the use of BDDs but still is on-the-fly.

1 Introduction

Model Checking [6,23] is a powerful technique for the verification of reactive systems. In particular, with the invention of symbolic model checking [4,22] very large systems, with more than 10^{20} states, could be verified. However, it is often observed, that explicit state model checkers [9,15] outperform symbolic model checkers, especially in the application domain of asynchronous systems and communication protocols [10]. We believe that the main reasons are the following: First, symbolic model checkers traditionally use binary decision diagrams (BDDs) [3] 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 to visit non reachable states without a prior reachability analysis.

In [2] 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 problem of symbolic model checking. Our main contribution is a new model checking algorithm that generalizes the tableau construction [1] 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 [16,17]. 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 [17] 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 [16,17].

Henzinger et. al. in [14] 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 [19], 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 [27], their result subsumes 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 [16], in the special case of FairCTL properties, to the tableau construction of [5]. However, this construction and also [14] do not allow the mixture of DFS and BFS, as in the layered approach of [17]. In addition, DFS was identified as one major reason for explicit state model checking 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 [14] requires two translations, from LTL to Büchi Automata and then to *post-*μ. A similar argument applies to [5,8]. Second it connects the local model checking paradigm of [1] with symbolic model checking in a natural way, thus combining BDD based with on-the-fly model checking. Finally our approach shows, that the idea of mixing DFS with BFS can be lifted from path expressions [17] to LTL.

Our procedure is correct and complete for all of LTL. If we consider existential model checking problems with no eventualities, then the size of the generated tableaux is linear in the number of states. Checking eventualities may result in an tableau with exponential size in the number of states. We are currently working on an extension that remains complete for all of LTL and produces tableaux with size linear in the number of states.

Our paper is organized as follows. In the next section our notation is introduced. Section 2 presents our new tableau construction. The following section considers an essential optimization, followed by a discussion of the complexity and the comparison with related work. 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 $\ell\colon \Sigma \to \wp(\mathcal{A})$ the labeling of the states with *atomic propositions*. As temporal operators we consider, the *next time* operator $\mathbf{X}$, the *finally* operator $\mathbf{F}$, the globally operator $\mathbf{G}$, the *until* operator $\mathbf{U}$, and its dual, the *release* operator $\mathbf{R}$. We use the standard semantics of CTL* as in [11]. We further assume the formulae to be in negation normal form, as in [1,7,8]. Thus negations only occur in front of atomic propositions. This restriction does not lead to an exponential blow up because we included the $\mathbf{R}$ operator that fulfills the property $\neg(f \ \mathbf{U}\ g) \equiv \neg f \ \mathbf{R}\ \neg g$.

3 Tableau Construction

In this section we present a new model checking algorithm for solving existential LTL model checking problems. In particular, given a Kripke structure K and an LTL formula f, the algorithm determines whether $\Sigma_0 \models \mathbf{E}f$, where $S \models \mathbf{E}f$ iff there exists a path $\pi \in \Sigma^\omega$ with $\pi(0) \in S$ and $\pi \models f$. A procedure for generating counterexamples, in case $\Sigma_0 \models \mathbf{E}f$ does not hold, is also included.

The algorithm is based on a tableau construction. Each tableau node is a sequent σ that contains a set of states $S \subseteq \Sigma$ and an LTL formula f (written $S \vdash \mathbf{E}(f)$). The rules for the construction of the tableau are very similar to those in [18], which is the dual construction of [1] for LTL with an existential path quantifier.

The main difference to [1,18] is also the main idea of our paper. We use sets of states instead of single states as one part of the sequent. With this modification we are able to represent set of states symbolically and use efficient BDD algorithms.

For the rest of the paper let $S \subseteq \Sigma$ be a set of states and $\mathbf{E}\Phi = \mathbf{E}\bigwedge \Phi_i$ be a conjunctively decomposed ELTL formula. We also use the notation $\mathbf{E}(\Phi, f)$ with the semantics $\mathbf{E}((\bigwedge \Phi_i) \wedge f)$. Further, for $S \subseteq \Sigma$, $p \in \mathcal{A}$, we define

$$S_p := \{s \in S \mid p \in \ell(s)\}, \qquad S_{\neg p} := \{s \in S \mid p \notin \ell(s)\}$$

$$\operatorname{Img}(S) := \{t \in \Sigma \mid \exists s \in S.\ (s,t) \in \delta\}$$

Given an initial set of states S (e.g. Σ_0) and an ELTL formula f we construct

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

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

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

$$R_{\mathbf{X}}\colon \frac{S \vdash \mathbf{E}(\mathbf{X}\Phi_1, \ldots, \mathbf{X}\Phi_n)}{\mathrm{Img}(S) \vdash \mathbf{E}(\Phi_1, \ldots, \Phi_n)} \qquad R_{\mathcal{A}^+}\colon \frac{S \vdash \mathbf{E}(\Phi, p)}{S_p \vdash \mathbf{E}(\Phi)}$$

$$R_{\mathrm{split}}\colon \frac{S \vdash \mathbf{E}(\Phi)}{S_1 \vdash \mathbf{E}(\Phi) \qquad S_2 \vdash \mathbf{E}(\Phi)}\ S_1 \cup S_2 = S \qquad R_{\mathcal{A}^-}\colon \frac{S \vdash \mathbf{E}(\Phi, \neg p)}{S_{\neg p} \vdash \mathbf{E}(\Phi)}$$

Fig. 1. Tableau rules.

a tableau by repeatedly applying the rules of Figure 1 starting with the root $S \vdash \mathbf{E}(f)$.

We continue the application of the rules until no new sequents can be added. In the resulting graph, which we call a tableau, every sequent occurs only once. Note that a tableau may be cyclic and, in general, is not uniquely defined.

Following [1] we first define a successful path in the tableau: A *finite* path through the tableau that ends with a sequent $S \vdash \mathbf{E}(\Phi)$ is called *successful* iff $S \neq \emptyset$ and $\Phi = \emptyset$. An *infinite* path X is called successful iff for every $\mathbf{F}g \in X(i)$, and every $f\ \mathbf{U}\ g \in X(i)$ there exists a $j \geq i$ with $g \in X(j)$.

A tableau is called successful if it contains a successful path. From this successful path we can construct a witness for the existential model checking problem associated with the root sequent of the tableau.

The following theorem shows that, no matter in which order we apply the tableau rules, the resulting tableau is successful iff the root sequent is satisfiable. We call a sequent $S \vdash \mathbf{E}(f)$ satisfiable iff $S \models \mathbf{E}f$.

Theorem 3.1 *Let K be a Kripke structure, $\mathbf{E}f$ an ELTL formula, and T a tableau with root $\Sigma_0 \vdash \mathbf{E}(f)$. Then $\Sigma_0 \models \mathbf{E}f$ iff T is successful.*

The proof consists of the combination of the following Lemma with the correctness and completeness results of [1,18]. We call a path x of sequents *singleton path* iff every sequent in x contains only a singleton set of states.

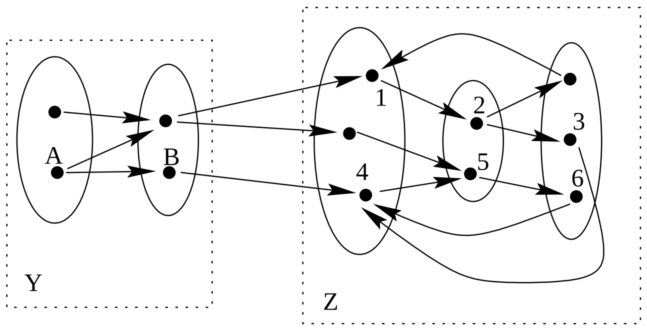

Fig. 2. Example for witness (resp. counterexample) generation.

Further let $X = (S_0 \vdash \mathbf{E}(f_0), S_1 \vdash \mathbf{E}(f_1), \ldots)$ be a finite or infinite path, then a singleton path $x = (\{s_0\} \vdash f_0, \{s_1\} \vdash f_1, \ldots)$ *matches* X iff $s_i \in S_i$ and if $X(i+1)$ is the result of applying $\mathbf{R_X}$ to $X(i)$, i.e. $X(i+1) = \mathrm{Img}(X(i))$, then $(s_i, s_{i+1}) \in R$.

Lemma 3.2 *Let X be a successful path for the root sequent $S \vdash \mathbf{E}(f)$. Then there exists $s \in S$ and a successful singleton path x for the root sequent $\{s\} \vdash \mathbf{E}(f)$ that matches X.*

The Lemma is proven by constructing a matching singleton path from a successful path. What follows is a sketch of this algorithm for an infinite path $X = Y \cdot Z^\omega$. A sequent σ is called an $\mathbf{X}$-sequent iff the $\mathbf{R_X}$ rule is applicable to σ, i.e. all formulae on the right hand side of σ are prefixed with the next time operator $\mathbf{X}$. For the purpose of constructing a singleton path only the $\mathbf{X}$-sequents of X are considered. We pick an arbitrary state s out of the first $\mathbf{X}$-sequent in Z. Note that s is also contained in $X(j+1)$ with $X(j)$ an $\mathbf{X}$-sequent and $j \geq |Y| + |Z|$.

Now we traverse the $\mathbf{X}$-sequents of Z until the last $\mathbf{X}$-sequent of Z is reached. During this traversal we choose an arbitrary successor state from the following $\mathbf{X}$-sequent. We can not choose a successor state in the immediate successor sequent, since this successor state might be eliminated by the application of the $R_\mathcal{A}$ rule before the next $\mathbf{X}$-sequent is reached. When the last $\mathbf{X}$-sequent in Z is reached then we check if the state chosen initially can be reached in one step from the current state. If this is the case, then we found a singleton cycle and continue to search a prefix singleton path for this cycle in Y.

Otherwise we repeat the traversal of Z, starting from an arbitrary image state of the last state, that is contained in the first $\mathbf{X}$-sequent of Z, until such a cycle is found. Because Σ and thus the number of different sequents is finite, the algorithm has to terminate. The resulting singleton path obviously matches the original path and is successful if the original path was successful. Consider the example of Figure 2 where each ellipsis depicts an $\mathbf{X}$-sequent.

The arrows between the single states are transitions of the Kripke structure. We start with 1, transition to 2 and pick 3 as successor of 2. The next transition, from 3 to 4, brings us back to the first **X**-sequent of Z but no cycle can be closed yet. We continue with 5 and 6 and finally reach 4 again. From there we find a prefix (A, B), that leads from the initial state A to the start of the cycle at 4. The resulting singleton path is $(\mathrm{A}, \mathrm{B}) \cdot (4, 5, 6)^\omega$. Note that this algorithm is actually used for the generation of a witness for the root formula (or a counterexample for the negation of the root formula).

The theorem follows by the observation that every successful singleton path can be interpreted as a successful path in the sense of [1,18] and vice versa. This mapping has to take into account the split rule R_{split} but otherwise just maps a singleton set into the single state contained in the set. Note that the tableaux for x and X, in general, are different.

For instance 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. The tableau for checking $\mathbf{EG}p$ looks as follows

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

$$\begin{tikzcd} \underset{p}{\textcircled{0}} \arrow[r, bend left] & \underset{p}{\textcircled{1}} \arrow[l, bend left] \end{tikzcd}$$

and the application of $\mathbf{R_X}$ to the leaf sequent leads back to the root sequent. The tableau represents one successful path that contains only one image calculation. However both matching singleton paths need two image computations before the loop can be closed:

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

Again the application of $\mathbf{R_X}$ to the leaf nodes yields the root. In general, matching singleton paths may require longer closing cycles than a matched path.

3.1 Algorithm

A more detailed description of the tableau construction is presented in this section. The overall approach expands open branches in DFS manner and stops when a successful path has been generated. In this case the formula can be fulfilled. If no successful path can be found and the tableau has been fully generated then the algorithm stops reporting that no witness has been found.

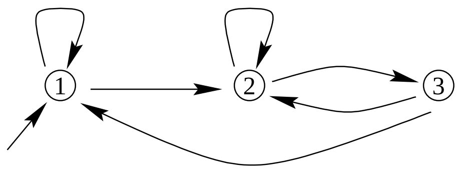

Fig. 3. Example Kripke structure.

If a leaf of a tableau is expanded and a sequent is generated that already occurred in the tableau then we found a successful path if the previous occurrence is on the path from the root to the expanded node and all eventualities on this path are fulfilled. If the new sequent occurs in the tableau but not on the path from the root to the expanded leaf, the parent of the new sequent, then we already have proven that the new sequent is unsatisfiable. In the remaining case, the new sequent occurs on the path from the root to the expanded node and at least one eventuality is not fulfilled, the strongly connected components of the tableau have to be considered, as in [1].

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. Hash tables work very well in practice.

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 contain $\mathbf{E}(\Phi)$. To check if a sequent already occurred, we just go through the list of corresponding formulae and check whether the BDDs representing the sets of states are the same. We can also combine several nodes on unsuccessful branches with the same formula by computing the disjunction of the BDDs. But keeping the BDDs separate results in a partitioning of the search space and hopefully results in small BDDs. Note that the same approach works for the optimization discussed in section 4 with the only modification that we check for non empty intersection instead of checking for equality.

3.2 Heuristics

The rule R_{split} is not really necessary 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.

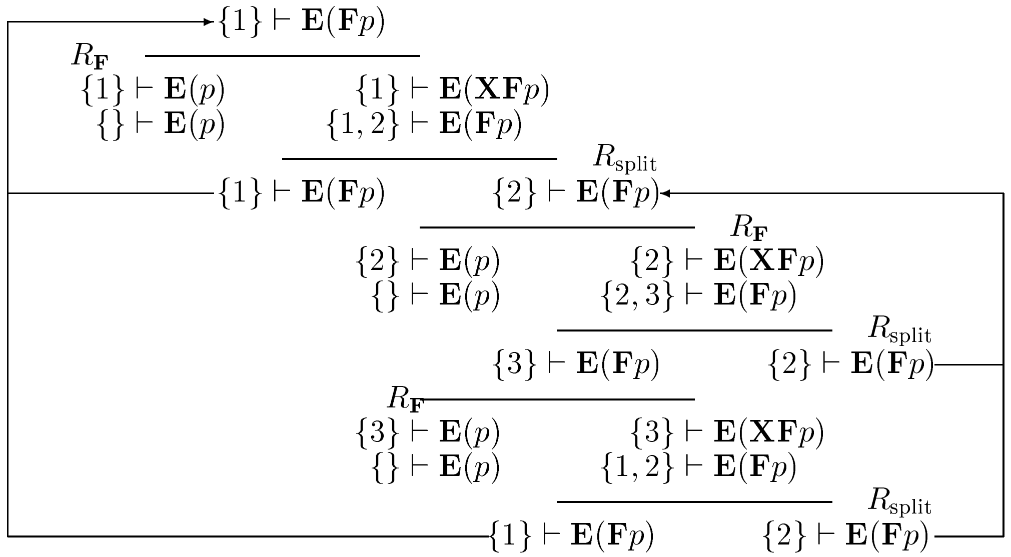
$\{1\} \vdash \mathbf{E}(\mathbf{F}p)$
$R_{\mathbf{F}}$
$\{1\} \vdash \mathbf{E}(p)$
$\{\} \vdash \mathbf{E}(p)$
$\{1\} \vdash \mathbf{E}(\mathbf{XF}p)$
$\{1,2\} \vdash \mathbf{E}(\mathbf{F}p)$
R_{split}
$\{1\} \vdash \mathbf{E}(\mathbf{F}p)$
$\{2\} \vdash \mathbf{E}(\mathbf{F}p)$
$R_{\mathbf{F}}$
$\{2\} \vdash \mathbf{E}(p)$
$\{\} \vdash \mathbf{E}(p)$
$\{2\} \vdash \mathbf{E}(\mathbf{XF}p)$
$\{2,3\} \vdash \mathbf{E}(\mathbf{F}p)$
R_{split}
$\{3\} \vdash \mathbf{E}(\mathbf{F}p)$
$\{2\} \vdash \mathbf{E}(\mathbf{F}p)$
$R_{\mathbf{F}}$
$\{3\} \vdash \mathbf{E}(p)$
$\{\} \vdash \mathbf{E}(p)$
$\{3\} \vdash \mathbf{E}(\mathbf{XF}p)$
$\{1,2\} \vdash \mathbf{E}(\mathbf{F}p)$
R_{split}
$\{1\} \vdash \mathbf{E}(\mathbf{F}p)$
$\{2\} \vdash \mathbf{E}(\mathbf{F}p)$

Fig. 4. Example for the usage of the split rule R_{split}.

Applying this heuristics essentially computes the set of reachable states in a BFS manner while checking on-the-fly for states violating the safety property. An example of this technique is shown in figure 4 using the Kripke structure of figure 3.

Another heuristic is to avoid splitting the tableau as long as possible. This is one of the heuristics proposed in [26] for the construction of small tableau as an intermediate step of translating LTL into the modal μ-calculus with the algorithm of [8]. In general, these heuristics are also applicable in our approach.

4 Optimization

The number of different left hand sides of sequents is exponential in $|\Sigma|$, the number of states of the Kripke structure. If we only consider LTL properties that do not contain eventualities, then we can apply an optimization that reduces the maximal number of different left hand sides, occurring in the tableau, to $|\Sigma| + 1$. This reduction can be achieved by modifying the tableau construction in such a way that all sequents with the same formula contain mutually exclusive set of states.

The tableau is built with DFS. The construction is stopped immediately if a successful path has been found. Otherwise the still open branches are expanded. If there are no more open branches the construction terminates with failure.

Assume that the result of applying a rule is a new sequent $\sigma = S \vdash \mathbf{E}(f)$ and there is another sequent $\sigma' = S' \vdash \mathbf{E}(f)$ with the same formula already in the tableau. First, if σ' is not on the path from the root to σ (this is a cross edge in terms of DFS), then we already have proven that all states $s \in S'$

can not fulfill $s \models \mathbf{E}f$. This allows us to remove all states in the intersection $S \cap S'$ and we use $S - S' \vdash \mathbf{E}(f)$ instead of σ as new tableau node.

Second let σ' be a predecessor of σ. Then we have to check if there is a self loop of a state in the intersection $S \cap S'$ along the segment. If this is the case a successful path has been found, since by our restriction the path does not contain any eventuality, and we can terminate the search immediately. Otherwise we can remove the intersection as in the previous case.

To check for a successful path, as in the last case, is similar to the generation of witnesses of Section 4. We start with the intersection $S \cap S'$ at σ' and compute all images along the segment restricting the image set to the set of states occurring in the sequents along the segment. If we reach σ and the set of states has become empty, then no loop is possible. This conclusion remains correct even if the path contains eventualities. Otherwise we repeat the calculation with the intersection of the calculated set with $S \cap S'$ restricting the images to previously calculated images. If we reach a fix point, a set that yields the same result after one iteration, then a successful path exists. A witness (resp. counterexample) can be extracted with the algorithm of Lemma 3.2.

If the optimization is applied without the restriction, i.e. the root formula contains eventualities, then our optimized procedure becomes incomplete, but the size of the tableau is linear in $|\Sigma|$. Incompleteness means, that a witness for an existential model checking problem, found by the optimized procedure, is indeed a witness. However if the procedure can not find a successful path, applying the optimization, then the root sequent might still be satisfiable.

5 Complexity and Related Work

In this section we discuss the complexity of our algorithm. Then we compare our approach 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 $\mathbf{O}(\exp(|\Sigma|) \cdot \exp(|f|))$. The time taken is polynomial in the size of the tableau. Thus the time complexity is (roughly) the same as the space complexity.

The optimization of the last section generates a tableau with the property that sequents with the same formula have mutually exclusive sets of states. Because there are no more than $|\Sigma|$ sets of states that are mutually exclusive, any formula occurs in at most $|\Sigma|$ sequents. Therefore the size of the resulting tableau is linear in the number of states and exponential in the size of the formula. Consequently our algorithm is polynomial in the number of states, with a small degree polynomial, and exponential in the size of the formula. However to achieve this complexity we have to restrict the class of properties or give up completeness.

This result almost matches the worst case complexity of explicit state model checking algorithms for LTL [20,1,15], which are linear in the number of states and exponential in the size of the formula. However, with our approach we are able to use efficient data structures to represent set of states symbolically and thus can hope to achieve exponentially smaller tableaux and exponentially smaller running times for certain examples.

The method of [8] translates an LTL formula into a tableau similar to the tableaux in our approach. In [8] the nodes contain only formulae and no states. 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 [12,21] 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 [14] an ELTL formula is translated to a Büchi automata with the method of [27]. This leads to an exponential blow up in the worst case. But see [13] 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 [12,21] can be derived. This translation produces a μ-calculus formula of alternation depth 2 which results in practically the same complexity as our algorithm.

The LTL model checking algorithm of [14] is also forward oriented. A forward state space traversal potentially avoids searching trough 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 [5] translates an LTL model checking problem into a FairCTL model checking problem. With the result of [12] 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. If [5] is not combined with [16,17], then it also shares the following disadvantage with the LTL model checking algorithm of [14]. The algorithm is based on BFS and it is not clear how to combine it DFS.

The work by Iwashita [16,17] does not handle full LTL and no complexity analysis is given. But if we restrict our algorithm to the path expressions of [16,17], then our algorithm subsumes the algorithms of [16,17], even for the *layered approach* of [17], the combination of DFS and BFS.

6 Conclusion

Although our technique clearly extends the work of [16,17] and bridges the gap between local and global model checking, we still need to show that it works in practice. In addition, a formalization of the optimization in Section 4 is necessary. We are also working on a complete tableau construction for eventualities with linear tableau size in the number of states. Finally, we want to investigate heuristics for applying the split rule. The approximation techniques of [24,25] are a good starting point.

References

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

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

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

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

[5] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *CAV'94*, LNCS. Springer, 1994.

[6] 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.

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

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

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

[10] 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.

[11] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited: on branching time versus linear time temporal logic. *Journal of the Association for Computing Machinery*, 33(1), 1986.

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

[13] 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.

[14] Thomas A. Henzinger, Orna Kupferman, and Shaz Qadeer. From Pre-historic to Post-modern symbolic model checking. In *CAV'98*, LNCS. Springer, 1998.

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

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

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

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

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

[20] 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.

[21] 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.

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

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

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

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

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

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