TECHNISCHE UNIVERSITÄT DRESDEN
Fakultät Informatik

Alternative Automata-based Approaches to Probabilistic Model Checking

Kurzversion der Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der Technischen Universität Dresden
Fakultät Informatik
von **David Müller**
geboren am 08. März 1987
in Dresden

Betreuende Hochschullehrerin:
Prof. Dr. rer. nat. Christel Baier
Technische Universität Dresden

Abstract

The goal of this thesis is the proposal of new methods for the analysis of Markov decision processes and Markov chains against LTL specifications. The standard approach transforms an LTL specification into a deterministic Rabin automaton which may result in a double-exponential blow-up.

We present two new approaches employing non-deterministic automata with a restricted form of non-determinism and one new approach employing deterministic automata but with a more flexible acceptance condition.

1 Introduction

The growing complexity and dependence on computational systems in our every day life renders checking their correctness and safety more complicate. Model checking provides a way to check correctness and safety by an exhaustive search of a model. In its classical form it decides whether a model of a system satisfies a property. It has been introduced in the early eighties of the last century, with the notable publications [EC80; LP85; CES86], and has spread out into different research lines by now, e.g., analysis of timed automata [AD94; LPY95; BY04] or probabilistic systems [Var85; VW86; CY95; Var99].

For expressing specifications such as safety one usually employs a temporal logic such computation tree logic or linear temporal logic (LTL) [Pnu77]. In this thesis we consider only LTL. It focuses on the paths of a model as the semantic of LTL is defined by a set of words. A model satisfies an LTL formula if all the model's traces are contained in the set of words satisfied by the LTL formula, i.e., the model exhibits only behavior characterized by the LTL formula. The typical approach for LTL model checking employs ω-automata. The negated LTL formula is transformed into a (possibly exponential-sized) non-deterministic Büchi automaton (NBA). Then a product is built, in which one can search for behavior that is forbidden by the specification.

In its simplest form model checking give binary answers like "The system obeys the specification" or "The system does not obey the specification". Probabilistic model checking (PMC) enables answers like "The system obeys the specification with a likelihood of 99.95%" by focusing on Markovian models like Markov chains or Markov decision processes (MDPs for short). Markov chains can be seen as labeled graphs equipped with a probability distribution on every state for choosing a successor state, and a probability distribution over all states, which serves to choose the initial state. If one provides non-deterministic choices to Markov chains, one obtains MDPs. In a game-based view MDPs can be called $1\frac{1}{2}$-player games, where one player resolves the non-deterministic choices, and the other (half) player resolves the probabilistic choices probabilistically. One distinguishes between qualitative and quantitative model checking in PMC. In qualitative PMC one wants to prove that a path property holds almost surely or with a positive probability in case of a Markov chain. On the other hand, in quantitative PMC one asks for the concrete probability that a path property holds.

The probabilistic choices of Markovian models hinders the direct employment of NBAs in the process of model checking Markovian models against LTL formula. As a resort to this problem, deterministic automata are usually employed.

The translation of LTL to deterministic ω-automata can cause a double-exponential blow-up [KV05; KR11]. This double-exponential blow-up together with the polynomial-time algorithms for building the product and the analysis of it yields an overall double-exponential-time algorithm for the analysis of Markov decision processes (MDP) against LTL. This double-exponential time algorithm matches the lower bound [CY95]. Nevertheless, in practical applications other approaches than via deterministic automata might increase the performance. In particular, limit-deterministic Büchi automata, these are automata behaving deterministically after an accepting state has been visited, can improve the performance for MDP analysis [Var85; KV15; Sic+16].

The case is different for Markov chains, where a PSPACE lower bound is known [Var85]. Thus, the double-exponential blow-up over deterministic automata leaves a complexity gap for Markov chains. In the literature there have been several approaches for Markov chain analysis with a single exponential runtime: an iterative refinement of the Markov chain [CY88; CY95], an approach via weak alternating automata [BRV04], and an approach over a special sub-class of unambiguous automata [CSS03].

The results in this thesis are based on [Kle+14; Bai+16; MS17] and unpublished material developed in a cooperation with Christel Baier, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, and Jan Strejček.

2 Markovian models, ω-automata and LTL

Markov decision processes. MDPs are an operational model for systems that exhibit non-deterministic and probabilistic choices [Bel57]. A Markov decision process is a tuple $\mathcal{M} = (S, Act, P, \iota, AP, \ell)$ where S is a finite set of states, Act a finite set of actions, $P : S \times Act \times S \to [0,1]$ is the transition probability function satisfying:

$$\sum_{s' \in S} P(s, \alpha, s') \in \{0, 1\} \qquad \text{for all } s \in S,\ \alpha \in Act,$$

$\iota : S \to [0,1]$ is the initial distribution satisfying $\sum_{s \in S} \iota(s) = 1$, and $\ell : S \to 2^{AP}$ is a labeling function. The size of an MDP $\mathcal{M}$, written as $|\mathcal{M}|$, is its number of states $|S|$.

Paths in $\mathcal{M}$ are finite or infinite sequences $\pi = s_0\, \alpha_0\, s_1\, \alpha_1\, s_2\, \alpha_2 \ldots$ starting in the initial state s_0 that are built by consecutive steps, i.e., $P(s_i, \alpha_i, s_{i+1}) > 0$ for all i. The trace of π is the word over the alphabet $\Sigma = 2^{AP}$ that arises by taking the projections to the state labels, i.e., $trace(\pi) = \ell(s_0)\, \ell(s_1)\, \ell(s_2) \ldots$. For an LTL formula φ over AP we write $\pi \models \varphi$ if $trace(\pi) \in \mathcal{L}(\varphi)$.

Strategies for the full player are called *schedulers*. In general, they can be history-dependent, i.e., a scheduler is a function $\mathfrak{s} \colon (S \times Act)^* \times S \to Act$ selecting the next action given the current path prefix. We call a path $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ an $\mathfrak{s}$-path if $\alpha_i = \mathfrak{s}(s_0, \alpha_0, s_1, \alpha_1, \ldots, \alpha_{i-1}, s_i)$ for all $i \geq 0$. Since the behavior of $\mathcal{M}$ is purely probabilistic if some scheduler $\mathfrak{s}$ is fixed, one can reason about the probability of path events. If L is an ω-regular language then $\Pr^{\mathfrak{s}}_{\mathcal{M}}(L)$ denotes the probability under $\mathfrak{s}$ for the set of infinite paths π with $trace(\pi) \in L$.

For a worst-case analysis of a system modeled by an MDP $\mathcal{M}$, one ranges over all schedulers (i.e., all possible resolutions of the non-determinism) and considers the

maximal or minimal probabilities for some ω-regular language L. Depending on whether L represents a desired or undesired path property, the quantitative worst-case analysis amounts to computing $\Pr^{\min}_{\mathcal{M}}(\varphi) = \min_{\mathfrak{s}} \Pr^{\mathfrak{s}}_{\mathcal{M}}(L)$ or $\Pr^{\max}_{\mathcal{M}}(L) = \max_{\mathfrak{s}} \Pr^{\mathfrak{s}}_{\mathcal{M}}(L)$. The existence of such schedulers is well-known, see, e.g., [Put94; FV96].

Markov chains. A special case of MDPs are discrete-time Markov chains (DTMCs for short), where $|Act(s)| = 1$ for every state $s \in S$. This results in the existence of exactly one scheduler, and therefore $\Pr^{\min}_{\mathcal{M}}$ and $\Pr^{\max}_{\mathcal{M}}$ coincides. We abbreviate the according probability by omitting min or max, i.e., $\Pr_{\mathcal{M}}$.

ω-automata. An ω-automaton $\mathcal{A} = (Q, \Sigma, \delta, Q_0, \Phi)$ is a tuple, where Q is a non-empty, finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the (non-deterministic) transition function, $Q_0 \subseteq Q$ is the non-empty set of initial states and Φ is the acceptance condition.

$\mathcal{A}$ is said to be *complete*, if $\delta(q, \sigma) \neq \varnothing$ for all states $q \in Q$ and all symbols $\sigma \in \Sigma$. $\mathcal{A}$ is called *deterministic*, if $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all $q \in Q$ and $\sigma \in \Sigma$. A *run* in $\mathcal{A}$ for an infinite word $w = \sigma_0 \sigma_1 \sigma_2 \ldots \in \Sigma^\omega$ is a sequence $\rho = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \ldots$ starting in an initial state q_0 such that $q_{i+1} \in \delta(q_i, \sigma_i)$ for all $i \in \mathbb{N}$. If the word w is clear, we sometimes omit the transitions and just write $\rho = q_0 q_1 \ldots$.

We write $\inf(\rho)$ to denote the set of all states occurring infinitely often in ρ. A run ρ is called accepting, if it meets the acceptance condition Φ, denoted by $\rho \models \Phi$. As the syntactical description of the acceptance condition we use the syntax presented in [Bab+15], where the acceptance condition is denoted by a positive Boolean combination of $\mathrm{Fin}(Z)$ or $\mathrm{Inf}(Z)$ atoms with $Z \subseteq Q$. We call this acceptance an Emerson-Lei acceptance.

The semantics of $\mathrm{Fin}(Z)$ and $\mathrm{Inf}(Z)$ are defined in straight-forward manner: A run $\rho = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \ldots$ is accepting for $\mathrm{Fin}(Z)$ if and only if $\inf(\rho) \cap Z = \varnothing$ holds, whereas ρ is accepting for $\mathrm{Inf}(Z)$ if and only if $\inf(\rho) \cap Z \neq \varnothing$ holds. $\mathrm{Fin}(\cdot)$ and $\mathrm{Inf}(\cdot)$ are dual to each other, i.e., every run ρ is accepting for $\mathrm{Inf}(Z)$ if and only if it is not accepting for $\mathrm{Fin}(Z)$, and analogously, ρ is accepting for $\mathrm{Fin}(Z)$ if and only if it is not accepting for $\mathrm{Inf}(Z)$.

We consider here the following four special types of acceptance conditions in particular and describe their constraints for infinite runs:

- *Büchi:* $\Phi = \mathrm{Inf}(Z)$ stands for a set of states, that needs to appear infinitely often.
- *parity:* Φ can be seen as a function $col : Q \to \mathbb{N}$ assigning to each state q a parity color and requiring that the least parity color appearing infinitely often is even. As formal syntax we fix $\mathrm{Inf}(Z_0) \vee (\mathrm{Fin}(Z_1) \wedge (\mathrm{Inf}(Z_2) \vee (\mathrm{Fin}(Z_3) \wedge \ldots)))$ with Z_i consisting of all states of color i.
- *generalized Rabin:* Φ is a disjunction of conjunctions, where every conjunction has at most one $\mathrm{Fin}(\cdot)$. Formally,

$$\Phi = \bigvee_{i \in \{1,\ldots,n\}} \left(\mathrm{Fin}(U_i) \wedge \bigwedge_{j \in \{1,\ldots,n_i\}} \mathrm{Inf}(L_{1,j}) \right),$$

i.e., requiring that for one of the conjunctions the states in U_i appear at most finitely often while in $L_{i,j}$ for every $j \in \{1, \ldots, n_i\}$ some state appears infinitely often. The term Rabin acceptance (without generalized) describes the special case where $n_i = 1$ for every i in $\{1, \ldots, n\}$

- *Streett:* Φ is dual to Rabin, i.e., it is a strong fairness condition. Syntactically, we fix $\bigwedge_{i \in \{1,\ldots,n\}} \mathrm{Fin}(U_i) \vee \mathrm{Inf}(L_i)$ as Streett acceptance.

Linear temporal logic (LTL). LTL extends Boolean logic by temporal operators such as $\bigcirc$ ("in the next step") and $\mathcal{U}$ ("until"). From $\mathcal{U}$ one can derive several operators as syntactic sugar, e.g., $\Diamond$ ("finally"), or $\Box$ ("globally"). The semantic of an LTL formula is defined by the infinite words satisfying it. For a deeper consideration, we refer to [BK08].

3 Good-for-games automata

A desire to avoid deterministic ω-automata occurred not only in the area of probabilistic model checking, but in the area of synthesis of reactive systems as well [KV05; KPV06; PPS06; SF07]. In 2006 Henzinger and Piterman [HP06] proposed the so-called good-for-games property for non-deterministic automata, a restricted form of non-determinism. This property has been independently proposed by Colcombet [Col09] for weighted automata, but here it is called history-deterministic. Henzinger and Piterman also developed an algorithm, which we call HP-algorithm, for constructing a good-for-games parity automaton out of an NBA, aimed at a compact symbolic representation.

In a good-for-games automaton, the non-determinism can be resolved in an incremental way for every accepted word without look-ahead. The formal definition of GFG ω-automata [HP06] relies on a game-based view of ω-automata. Given a complete ω-automaton $\mathcal{A}$ as before, we consider $\mathcal{A}$ as the game arena of an infinite, turn-based 2-player game, called *monitor game*: if the current state is q, then player 1 chooses a symbol $\sigma \in \Sigma$ whereas the other player (player 0) has to answer by a successor state $q' \in \delta(q, \sigma)$, i.e., resolves the non-determinism. In the next round q' becomes the current state. A *play* is a maximal alternating sequence $\varsigma = q_0\, \sigma_0\, q_1\, \sigma_1\, q_2\, \sigma_2 \ldots$ of states and (action) symbols in the alphabet Σ starting with an initial state q_0. Intuitively, the σ_i's are the symbols chosen by player 1 and the q_i's are the states chosen by player 0 in round i. Player 0 wins the play ς if whenever $\varsigma|_\Sigma = \sigma_0\, \sigma_1\, \sigma_2 \ldots \in \mathcal{L}_\omega(\mathcal{A})$ then $\varsigma|_Q = q_0\, q_1\, q_2 \ldots$ is an accepting run. A strategy for player 0 is a function $\mathfrak{f} : (Q \times \Sigma)^* \to Q$ with $\mathfrak{f}(\ldots q\, \sigma) \in \delta(q, \sigma)$ and $\mathfrak{f}(\varepsilon) \in Q_0$. A play $\varsigma = q_0\, \sigma_0\, q_1\, \sigma_1\, q_2 \ldots$ is said to be $\mathfrak{f}$-*conform* or an $\mathfrak{f}$-play if $q_i = \mathfrak{f}(q_0\, \sigma_0 \ldots \sigma_{i-2}\, q_{i-1} \sigma_{i-1})$ for all $i \geq 1$. An automaton $\mathcal{A}$ is called *good-for-games* if there is a strategy $\mathfrak{f}$ such that player 0 wins each $\mathfrak{f}$-play. Such strategies will be called *GFG-strategies* for $\mathcal{A}$. Obviously, each complete deterministic automaton enjoys the GFG property.

MDP analysis. We address the task to compute the maximal or minimal probability in an MDP $\mathcal{M}$ for the path property imposed by a non-deterministic ω-automaton $\mathcal{A}$.

In the context of MDP analysis, we always assume w.l.o.g. that a good-for-games automaton has exactly one initial state. For a good-for-games automaton with several initial states, we pick an arbitrary GFG strategy $\mathfrak{f}$, and set the unique initial state q_0 to the state $\mathfrak{f}(\varepsilon)$ chosen by $\mathfrak{f}$ for the empty word.

We now modify the standard definition of the product of an MDP with a non-deterministic ω-automata (with a unique initial state). The crucial difference is that the actions are now pairs $\langle \alpha, p \rangle$ consisting of an action in $\mathcal{M}$ and a state in $\mathcal{A}$, representing the non-deterministic alternatives in both the MDP $\mathcal{M}$ and the automaton $\mathcal{A}$. Formally, let $\mathcal{M} = (S, Act, P, \iota, AP, \ell)$ be an MDP and $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Phi)$ a complete non-deterministic ω-automaton with $\Sigma = 2^{AP}$. The product MDP is

$$\mathcal{M} \otimes \mathcal{A} \;=\; (S \times Q, Act \times Q, P', \iota', Q, \ell')$$

where the transition probability function P' is given by $P'(\langle s, q \rangle, \langle \alpha, p \rangle, \langle s', q' \rangle) = P(s, \alpha, s')$ if $p = q' \in \delta(q, \ell(s))$. In all other cases $P'(\langle s, q \rangle, \langle \alpha, p \rangle, \langle s', q' \rangle) = 0$. The initial distribution is given by $\iota'(\langle s, q \rangle) = \iota(s)$ if $q = q_0$ and $\iota'(\langle s, q \rangle) = 0$ in all other cases.

In the product, the states of the automaton serve as the atomic propositions and the labeling function is given by $\ell'(\langle s, q \rangle) = \{q\}$. This allows us to consider the traces in $\mathcal{M} \otimes \mathcal{A}$ simply as words over the alphabet Q. Likewise, $\mathcal{A}$'s acceptance condition Φ can be seen as a language over Q, which permits treating Φ as a property for the paths in $\mathcal{M} \otimes \mathcal{A}$. In particular, for $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi)$, Φ corresponds to the set of paths in the product where the projection on the $\mathcal{A}$-states yields an accepting path in $\mathcal{A}$.

Theorem 3.1. *For each MDP $\mathcal{M}$ and non-deterministic ω-automaton $\mathcal{A}$ as above:*

(a) $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi) \;\leq\; \Pr^{\max}_{\mathcal{M}}(\mathcal{A})$

(b) *If $\mathcal{A}$ is good-for-games, then:* $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\Phi) \;=\; \Pr^{\max}_{\mathcal{M}}(\mathcal{A})$

The computation of maximal probabilities for properties given by a standard ω-regular acceptance condition Φ (e.g., Büchi, Rabin, parity or Streett) can be carried out by a graph analysis that replaces Φ with a reachability condition and linear programming techniques for computing maximal reachability probabilities. See, e.g., [BK08; BGC09a; BGC09b]. The time complexity is polynomial in the size of $\mathcal{M}$ and $\mathcal{A}$, matching the complexity of the approach using deterministic ω-automata.

GFG automata generation. For the generation of GFG automata, we rely here on a modified version of Safra's determinization algorithm described in [HP06]. As improvement, we consider three heuristics: a loose variant, where some constraints on the state-space of the automaton are loosened (see [HP06]), an iterative approach, where the automaton is constructed incrementally, and a union heuristic similar to [KB06].

Additionally, we can show by a straight-forward adaption of [KV05] that similar to deterministic automata, a double-exponential blow-up in the translation from LTL to GFG automata is unavoidable in the worst-case.

Experimental evaluation. We summarize here on a number of experiments on the GFG automaton generation as described in [HP06] and the actual GFG-based MDP analysis. For the generation of GFG automata we use our implementation `ltl2gfg`, which reads an LTL formula as input, calls `ltl2ba` for NBA generation, and then applies the GFG automata construction. For the implementation of GFG-based MDP analysis we extended the symbolic `MTBDD` engine in `PRISM`.

For benchmarking the GFG automata generation itself, we took 94 benchmark formulas from [EH00; SB00; DAC99]. All our experiments were carried out on a computer with 2 Intel E5-2680 8-core CPUs at 2.70 GHz with 384GB of RAM running Linux and with a memory limit of 10 GB and a time-out of 30 minutes for each formula.

For every automaton we report on the number of BDD nodes in the encoding of the transition function, as this is the most crucial aspect.

To allow a fair comparison with the explicit determinization in `ltl2dstar`, we consider symbolic encodings of the DRA obtained from `ltl2dstar` 0.5.1, using the same LTL-to-NBA translator, i.e., `ltl2ba`, as used by `ltl2gfg`.

`ltl2dstar` did not get a timeout once, whereas `ltl2gfg` did receive a timeout between 20 (variant loose, iterative, union, and dynamic variable ordering enabled) and 45 (standard variant with dynamic variable ordering enabled) times. In a comparison of the symbolic representation (the number of BDD nodes needed for the transition function in particular) `ltl2gfg` in its standard variant had the worst performance, and `ltl2gfg` with all optimizations enabled had the best performance among all `ltl2gfg` variants. E.g., the number of automata with less than 100 BDD nodes has been 6 for `ltl2gfg` standard, 32 for `ltl2gfg` with all optimizations enabled, and 65 for `ltl2dstar`.

Implementation and experiments in `PRISM`. Despite the negative results of our experiments for the generation of GFG automata, we investigated the use of good-for-games automata in the context of probabilistic model checking. It could be the case that particularities of the symbolic encoding or of the automaton's structure turn out to be beneficial in this setting.

We extended the MTBDD-based, symbolic engine of `PRISM` 4.1 with an implementation of our algorithm for computing $\Pr^{\max}_{\mathcal{M}}(\varphi)$ using GFG automata for φ (and $\Pr^{\min}_{\mathcal{M}}(\varphi)$ using a GFG automaton for $\neg\varphi$).

We compare this approach with the standard approach of `PRISM`, where an explicit DRA is constructed with an integrated version of `ltl2dstar`, which is then symbolically encoded as described before.

The analysis is then carried out symbolically by the `MTBDD` engine for every approach, i.e., where the matrix and the value vector are stored via MTBDDs.

If `PRISM` normally handles a formula via a specialized algorithm for simple path formulas that does not need an automata product construction, we forced the use of the general, automata-based approach. We have carried out our experiments with the different variants for the generation of GFG automata of `ltl2gfg`. As before, we impose a 30 minute time and 10 GB memory limit.

As case-study, we use a `PRISM` model [KNS02] from the `PRISM` benchmark suite for

parts of the WLAN carrier-sense protocol of IEEE 802.11. For details on the model we refer to `http://www.prismmodelchecker.org/casestudies/wlan.php`

It models a two-way handshake mechanism of the IEEE 802.11 (WLAN) medium access control scheme with two senders that compete for the medium. As messages get corrupted when both senders send at the same time (called a collision), a probabilistic back-off mechanism is employed to reduce the likelihood of collisions. The back-off procedure is the key feature of the protocol, which is started if an error occurred or the sender wants to send a new message after sending a message. The back-off procedure consists of waiting a randomized amount of time while the channel has to be free. It ends with retrying to send a message. To define the maximal amount of waiting time in the back-off procedure, the model is parametrized by the parameter `MAX_BACKOFF`. Here, we consider the values `MAX_BACKOFF` $\in \{1, \ldots, 6\}$. Since stations cannot listen to their own transmissions, after having started to transmit a message, they cannot determine for a short amount of time whether another station has started to send at the same moment, called the *vulnerable* section. To reduce the likelihood of these collisions, a station has to check that the channel is free for a fixed time period. This happens in state `Wait_Difs`. In the model, a collision counter is used to record the number of collisions for use in the formulas. For our experiments, we set the maximum number of collisions that can be counted to 4.

For this benchmark we consider the following LTL path properties:

In our tables, we refer by `WLAN`n to the case-study MDP with maximum back-off value `MAX_BACKOFF`, for n from 1 to 6.

To provide an overview of the behavior of the different variants of `ltl2gfg` in the context of `PRISM`, Table 1 compares the running time of some variants against the baseline of the DRA-based standard approach. We consider the 36 combinations of `WLAN1` to `WLAN6` and φ_1 to φ_6. The table lists the number of cases where a timeout occurred and where time spent using the GFG approach exceeded the standard approach only by a given factor. We refer to the baseline time spent using the standard approach in `PRISM` as t_{STD} and to the time spent using the GFG approach (when there was no timeout) as t_{GFG}. For example, if we consider the loose variant with active iterative approach, in 11

	std.	loose	loose dyn.	loose union	loose it.	loose it. dyn.	loose it. union
$t_{\text{GFG}} < 3 \cdot t_{\text{STD}}$	7	8	7	7	7	7	9
$t_{\text{GFG}} < 7 \cdot t_{\text{STD}}$	11	11	11	15	16	13	15
$t_{\text{GFG}} < 20 \cdot t_{\text{STD}}$	17	17	17	21	22	21	21
$t_{\text{GFG}} < 250 \cdot t_{\text{STD}}$	29	27	29	30	32	28	30
$t_{\text{GFG}} \leq 30m$	31	28	30	32	33	32	32
Aborted	5	8	6	4	3	4	4

Table 1: Results for model checking IEEE 802.11 with `PRISM` and different variants of `ltl2gfg`.

of the 36 cases the running time of `PRISM` with the GFG approach was within the time spent by the standard `PRISM` approach multiplied by the factor 7.

As it was to be expected given our results on the automata construction, the GFG-based analysis did not improve on the standard approach. Even using the optimal variant of `ltl2gfg` for each formula, ignoring the automata construction times, and for cases where the product $\mathcal{M} \otimes \mathcal{A}$ had a comparable BDD size for the GFG- and DRA-based approach, the model checking using the GFG automata took significantly longer.

We also evaluated our GFG-based approach on a case-study for the dining philosopher's problem [LSS94]. The benchmarks for this case-study reflects the results of the case-study for the IEEE802.11 protocol.

4 Unambiguous automata

Unambiguity is a widely studied generalization of determinism with many important applications in automata-theoretic approaches, see e.g. [Col12; Col15]. A non-deterministic automaton is said to be unambiguous if each word has at most one accepting run. In this thesis we consider unambiguous Büchi automata (UBA) over infinite words. Not only are UBA as expressive as the full class of NBA [Arn85], they can also be exponentially more succinct than deterministic automata. LTL formulas can be translated into UBA with a single-exponential blow-up at most, just as NBA.

In the context of PMC, unambiguous automata over finite words provide an elegant approach to compute the probability for a regular safety or co-safety property in finite-state Markov chains [BLW14]. A generalization to UBA has been proposed in [BLW13; BLW14], but the presented approaches are flawed. The flaw lies in the assumption that a Markov chain state s and UBA state q with $\Pr_{\mathcal{M}[s]}(\mathcal{A}[q]) = 1$ exists if and only if $\Pr_{\mathcal{M}}(\mathcal{A}) > 0$.

As remedy to this flaw we consider every SCC in the Markov chain/UBA product and determine whether it contains a state (s, q) inducing a positive probability. This holds, if and only if the transition matrix of the SCC has not full rank. If the SCC contains a state (s, q) inducing a positive probability, we evaluate the probability $\Pr_{\mathcal{M}[s]}(\mathcal{A}[q])$ for every product state (s, q) in the SCC. To calculate these probabilities, we rely on a set of product states inducing probability 1. Such a set can be efficiently constructed in polynomial time if there exists such a set.

After we have calculated the probabilities within every SCC containing a state inducing positive probability we derive the probabilities for the remaining states in polynomial time. Our overall result is as follows:

Theorem 4.1. *Given a Markov chain $\mathcal{M}$ and a UBA $\mathcal{U}$, the value $\Pr_{\mathcal{M}}(\mathcal{L}_\omega(\mathcal{U}))$ is computable in time polynomial in the sizes of $\mathcal{M}$ and $\mathcal{U}$.*

UBA generation. For the transformation of LTL into UBA we employ `SPOT`, that generates UBA by an exhaustive application of the following rewrite rules:

$$\varphi \vee \psi \mapsto \varphi \vee (\neg\varphi \wedge \psi) \quad \varphi \,\mathcal{U}\, \psi \mapsto \psi \vee (\neg\psi \wedge \varphi \wedge \bigcirc(\varphi \,\mathcal{U}\, \psi) \quad \varphi \,\mathcal{R}\, \psi \mapsto \psi \wedge (\varphi \vee (\neg\varphi \wedge \bigcirc(\varphi \,\mathcal{R}\, \psi))$$

The idea behind these rules is that if one has possible non-deterministic branching, e.g., by a disjunction, the successors in the automaton should recognize disjoint languages.

Experimental evaluation. Our implementation of the Markov chain analysis is based on the `explicit` engine of `PRISM`, where the Markov chain is represented explicitly. Our implementation supports UBA-based model checking for handling LTL using an external LTL-to-UBA translator as well as direct verification against a path specification given by a UBA provided in the HOA format [Bab+15]. For the purpose of the benchmarks we employ the `ltl2tgba` tool from `SPOT` [Dur14] version 2.5 to generate a UBA for a given LTL formula.

For the linear algebra parts of the algorithms, we rely on the `COLT` library [Hos04].

Positivity Check. In the first analysis step, we check whether an SCC contains a state pair (s, q) such that $\Pr_{\mathcal{M}[s]}(\mathcal{L}_\omega(\mathcal{A}[q])) > 0$. We can rely on two different methods: On the one hand, we can compute the rank of the induced matrix as indicated above, on the other hand we can rely on an iterative matrix vector multiplication, which terminates if we reach a fixed point or obtain an vector, that is strictly smaller in every component than its predecessor. As the iterative matrix vector multiplication outperforms the rank computation, the following benchmark uses the matrix vector multiplication as positivity check.

Case Study: Bounded Retransmission Protocol. We report here on benchmarks using the bounded retransmission protocol (BRP) case study of the `PRISM` benchmark suite [KNP12]. The model from the benchmark suite covers a single message transmission, retrying for a bounded number of times in case of an error. We have slightly modified the model to allow the transmission of an infinite number of messages by restarting the protocol once a message has been successfully delivered or the bound for retransmissions has been reached. We will include benchmarks with pre-generated automata, as well as benchmarks with LTL as starting point.

Automata based specifications. We consider the property "the message was retransmitted k steps before an acknowledgment." To remove the effect of selecting specific tools for the LTL-to-automaton translation, we consider directly model checking against automata specifications at first. We use the pre-generated minimal DBA and UBA (denoted by $\mathcal{A}^k$ in Table 2) and an additional variant of them (denoted by $\mathcal{B}^k$), namely enforcing an infinite loop of retransmissions of "k steps before an acknowledgment".

Table 2 shows results for selected k (with a timeout of 30 minutes), demonstrating that for this case study and properties our UBA-based implementation is generally competitive with the standard approach of `PRISM` relying on deterministic automata. For $\mathcal{A}^k$, our implementation detects that the UBA has a special shape where all final states have a true-self loop which allows skipping the SCC handling. Without this optimization, t_{Pos} are in the sub-second range for all considered $\mathcal{A}^k$. At a certain point, the implementation of the standard approach in `PRISM` becomes unsuccessful, due to `PRISM` size limitations in

	PRISM standard			PRISM UBA			
	$\lvert\mathcal{A}^k_{DRA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}^k_{DRA}\rvert$	t_{MC}	$\lvert\mathcal{A}^k_{UBA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}^k_{UBA}\rvert$	t_{MC}	t_{Pos}
$k=4$, $\mathcal{A}^4$	33	61,025	0.4 s	6	34,118	0.3 s	
$\mathcal{B}^4$	33	75,026	0.4 s	6	68,474	1.3 s	1.0 s
$k=6$, $\mathcal{A}^6$	129	62,428	0.5 s	8	36,164	0.2 s	
$\mathcal{B}^6$	129	97,754	0.5 s	8	99,460	1.7 s	1.3 s
$k=8$, $\mathcal{A}^8$	513	64,715	0.6 s	10	38,207	0.3 s	
$\mathcal{B}^8$	513	134,943	0.7 s	10	136,427	2.6 s	2.1 s
$k=14$, $\mathcal{A}^{14}$	32,769	83,845	4.2 s	16	44,340	0.3 s	
$\mathcal{B}^{14}$	32,769	444,653	4.9 s	16	246,346	6.8 s	6.1 s
$k=16$, $\mathcal{A}^{16}$	131,073	–	–	18	46,390	0.3 s	
$\mathcal{B}^{16}$	131,037	–	–	18	282,699	8.9 s	8.0 s
$k=48$, $\mathcal{A}^{48}$	–	–	–	50	79,206	0.8 s	
$\mathcal{B}^{48}$	–	–	–	50	843,414	72.4 s	70.3 s

Table 2: Statistics for DBA/DRA- and UBA-based model checking of the BRP case study (parameters $N=16$, $MAX=128$), a DTMC with 29358 states, depicting the number of states for the automata and the product and the time for model checking (t_{MC}). For $\mathcal{B}$, the time for checking positivity (t_{Pos}) is included in t_{MC}. The mark – stands for "not available" or timeout (30 minutes).

the product construction of the Markov chain and the deterministic automaton ($\mathcal{A}^k/\mathcal{B}^k$: $k \geqslant 16$). As can be seen, using the UBA approach we were able to successfully scale the parameter k beyond 48 when dealing directly with the automata-based specifications ($\mathcal{A}^k/\mathcal{B}^k$) and within reasonable time required for model checking.

LTL based specifications. We consider here two LTL properties: The first one is:

$$\varphi^k = (\neg\texttt{ack\_received})\ \mathcal{U}\ \big(\texttt{retransmit} \wedge (\neg\texttt{ack\_received}\ \mathcal{U}^{=k}\ \texttt{ack\_received})\big),$$

where $a\,\mathcal{U}^{=k}b$ stands for $a \wedge \neg b \wedge \bigcirc(a \wedge \neg b) \wedge \ldots \wedge \bigcirc^{k-1}(a \wedge \neg b) \wedge \bigcirc^k b$. The formula φ^k ensures that k steps before an acknowledgment the message was retransmitted. Hence, it is equivalent to the property described by the automaton $\mathcal{A}^k$. For the LTL-to-automata translation we included the Java-based PRISM reimplementation of `ltl2dstar` [KB06] to obtain a deterministic Rabin automaton (DRA) for the standard PRISM approach. For the generation of UBA, we relied on SPOT, as it is the only tool that is capable of generating UBA explicitly. SPOT actually generates a UFA for φ^k which is recognized by our implementation in PRISM as explained above.

Table 3 lists the results for model checking φ^k. From a certain point on, the implementation of the standard approach in PRISM is unsuccessful, due to PRISM restriction in the DRA construction ($k \geqslant 8$) or a timeout during UBA construction ($k \geqslant 13$).

SPOT produces unnecessary large UBA for φ^k. This is also reflected in the automata generation times, as for $k \geq 13$, SPOT was not able to produce a UBA within the time bound of 30 min.

k	PRISM standard			PRISM UBA		
	$\lvert\mathcal{A}_{DRA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{DRA}\rvert$	t_{MC}	$\lvert\mathcal{A}_{UBA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{UBA}\rvert$	t_{MC}
4	122	29,358	0.6 s	21	34,757	0.3 s
6	4,602	29,358	1.8 s	71	37,951	0.4 s
8	–	–	–	265	41,902	1.1 s

Table 3: Statistics for UBA based model checking of the BRP model and φ^k in comparison to the standard method. For every approach except the corresponding automata sizes, product sizes are depicted and the overall model checking times (t_{MC}) are listed, which includes the time for automata translation.

k	PRISM standard			PRISM UBA				
	$\lvert\mathcal{A}_{DRA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{DRA}\rvert$	t_{MC}	$\lvert\mathcal{A}_{UBA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}_{UBA}\rvert$	t_{pos}	t_{Cut}	t_{MC}
1	6	29,358	0.3 s	4	31,422	<0.1 s	n/a	0.3 s
2	17	37,678	0.4 s	8	41,822	4.5 s	0.2 s	5.0 s
3	65	39,726	0.4 s	14	45,934	4.9 s	0.2 s	5.5 s
4	314	43,806	0.5 s	22	54,126	5.6 s	0.2 s	6.2 s
5	1,443	47,902	1.1 s	32	62,334	6.4 s	0.2 s	7.0 s
6	9,016	56,029	3.9 s	44	78,669	9.1 s	0.3 s	9.9 s
7	67,964	–	–	58	86,853	10.1 s	0.4 s	11.0 s
8	–	–	–	74	103,157	13.4 s	0.2 s	14.4 s
10	–	–	–	112	127,562	19.2 s	0.2 s	21.2 s

Table 4: Statistics for UBA based model checking of the BRP model and ψ^k in comparison with to the standard method. The structure of this table corresponds to Table 3, but with additional listing of the time for the positivity checks t_{pos} and cut calculation time t_{cut}. n/a means not available.

As second formula we investigate

$$\psi^k = \Box(\texttt{msg\_send} \rightarrow \Diamond(\texttt{ack\_send} \wedge \Diamond^{\leqslant k}\texttt{ack\_received})),$$

where $\Diamond^{\leqslant k}a$ denotes $a \vee \underbrace{\bigcirc(a \vee \bigcirc(\ldots \vee \bigcirc a))}_{k \text{ times}}$. This formula requires that every request (sending a message and waiting for an acknowledgment) is eventually responded by an answer (the receiver of the message sends an acknowledgment and this acknowledgment is received within the next k steps).

Table 4 summarizes the result of the benchmark for ψ^k. Here, the `PRISM` standard approach with its own implementation of `ltl2dstar` was able to finish the calculations until $k = 6$. For $k = 7$, `PRISM` standard was able to construct the DRA (with 67,964 states and within 27.5 seconds), but not able to construct the product anymore.

In contrast to the deterministic automata, the UBA sizes increase moderately for an increasing k. In the UBA approach the positivity check is the most time-consuming part of the calculation. For $k = 1$ there is no positive SCC, so the cut calculation is omitted. The model checking process consumes more time in the UBA case in comparison with `PRISM` standard until $k = 6$, but for bigger k the performance turned around. Even if `PRISM` standard had completed the calculation for $k = 7$, it would have been slower as the creation of the DRA took 27.5 seconds.

4.1 NBA vs UBA

To gain some understanding on the cost of requiring unambiguity for an NBA, we have compared the sizes of NBA and UBA generated by the `ltl2tgba` tool of `SPOT` for the formulas of [EH00; SB00; DAC99] used for benchmarking, e.g., in [KB06]. We consider both the "normal" formulas and their negations, yielding 188 formulas.

Number of states $\leq x$	≤ 1	≤ 2	≤ 3	≤ 4	≤ 5	≤ 7	≤ 10	≤ 12	≤ 20	> 20
`ltl2tgba` NBA	12	49	103	145	158	176	181	185	188	0
`ltl2tgba` UBA	12	42	74	108	123	153	168	173	180	8

Table 5: Number of formulas where the (standard) NBA and UBA has a number of states $\leq x$.

As can be seen in Table 5, both the NBA and UBA tend to be of quite reasonable size. Most of the generated UBA (102) have the same size as the NBA and for 166 of the formulas the UBA is at most twice the size as the corresponding NBA. The largest UBA has 112 states, the second largest has 45 states.

5 Emerson-Lei acceptance

In contrast to the two previous sections, where we considered restricted forms of nondeterminism, we reexamine here the Muller acceptance, but we add a crucial twist by a symbolic representation.

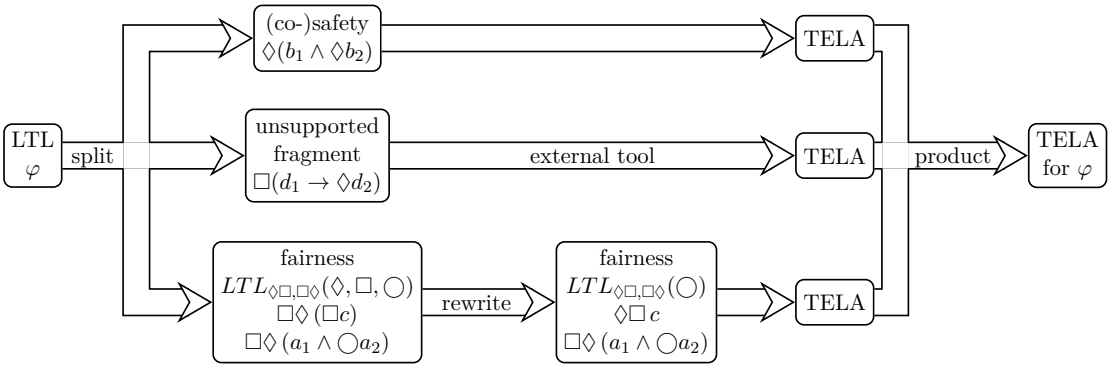

Figure 1: The input LTL formula is split up, each subformula is translated independently, and then a product automaton is constructed, as can be seen for the example $\varphi = \square\Diamond(a_1 \wedge \bigcirc a_2) \wedge \Diamond(b_1 \wedge \Diamond b_2) \wedge \square\Diamond(\square c) \wedge \square(d_1 \rightarrow \Diamond d_2)$. The abbreviation TELA stands for transition-based Emerson-Lei automata.

We call this new symbolic Muller acceptance Emerson-Lei acceptance. It is a positive Boolean formula with $\mathrm{Fin}(Z)$ and $\mathrm{Inf}(Z)$ as atoms, where $Z \subseteq Q$ is an arbitrary subset of the state-space in case of a state-based acceptance, and $Z \subseteq Q \times \Sigma \times Q$ in case of a transition-based acceptance.

A lot of research has been put into the translation of LTL to deterministic ω-automata, but to the best of our knowledge all those translations are targeted at a particular acceptance like Rabin, Streett or parity. There exists already the idea of a production construction in [Bab+15] to obtain a complex acceptance: The formula is split into several subformulas, where the top-most operator in the syntax tree is a temporal operator, and the subformulas are not nested within the scope of another temporal operator. These subformulas are converted to deterministic automata with one of the already known translators. These deterministic automata are composed in a product construction, where the graph is the standard product, and the acceptance reflects the structure of the original formula, where every top-most temporal formula is replaced by the acceptance condition of the corresponding automaton.

We consider special fairness properties in particular and give a translation based on buffers. For safety and co-safety LTL formulas we rely on the *af* function [EKS16; Sic+16] computing the left-derivative directly on the formula. Additionally, if we encounter a subformula not contained in our supported fragments for a direct translation, we rely on external tools for translation, and compose a deterministic automaton for the overall formula with the product construction of [Bab+15]. We improve the product construction and make use of the knowledge about the subformulas, if one subformula falls into the fairness or co-safety/safety fragment. A general scheme for our approach is depicted in Figure 1, which we implemented in our tool `Delag` (Deterministic Emerson-Lei Automata Generator).

As a second theoretical contribution we consider the complexity of deciding $\Pr^{\max}_{\mathcal{M}}(\Phi) >$

0 for an MDP $\mathcal{M}$ and an Emerson-Lei acceptance Φ. We prove, that the positivity problem is NP-complete for a class of Emerson-Lei acceptances, if the satisfiability problem is NP-complete for the corresponding class of Boolean formulas. This NP-completeness result inspired us to take on a well-known SAT-solving algorithm, called DPLL (see [DP60] and its refinement [DLL62]). This DPLL-based algorithm for deciding positivity for the whole class of Emerson-Lei acceptance turns out to be polynomial in time for Streett and (generalized) Rabin acceptance.

Fairness fragment. The fairness fragment consists of LTL formulas, where only $\bigcirc$, $\Diamond$, or $\Box$ occur as temporal operators, and $\Diamond\Box$ or $\Box\Diamond$ as temporal operators on the top-most position in the syntax tree. By known LTL rewrite rules we can ensure that every formula is written into a normal form where the formulas have on the top-most position in the syntax tree a $\Diamond\Box$ or a $\Box\Diamond$ and the only further allowed temporal operator is $\bigcirc$.

This normal form enables us to provide for each fairness formula an equivalent automaton with a single acceptance set (either $\mathrm{Fin}(P)$ or $\mathrm{Inf}(P)$), where we use the direct correspondence between $\Diamond\Box\varphi$ and $\mathrm{Fin}(P)$ for a suitable set of transitions P, and $\Box\Diamond\varphi$ and $\mathrm{Inf}(P)$. If the formula does not contain any $\bigcirc$, then we can even provide a one-state automaton.

Safety fragment. The safety fragment describes ω-regular languages for which there exists a set of so-called bad prefixes. Every infinite word with such a bad prefix does not belong to the safety language. Complementary, for a co-safety language there exists a language of good prefixes, such that after consuming a good prefix the infinite word belongs to the language, independently of the infinite suffix. The translation of the (co-)safety fragments is well-known and we rely on the construction of `Rabinizer` (see, e.g., [EKS16]). The resulting automaton of a co-safety language contains a unique state with an accepting *true* self-loop, whereas the automaton of a safety language contains a unique rejecting state with a rejecting *true* self-loop.

Product automaton. Each individual automaton for every subformula is joined via a product construction. The standard product construction runs every automaton synchronously and yields an automaton in the size of the product of all automata sizes in the worst case.

As this synchronous tracking of automata states may not be necessary, we introduced some enhancements in the standard product. For this, we introduced some extra states per automaton with a fixed semantics: q_{acc} is a state which accepts every infinite word, q_{rej} is a state which rejects every infinite word, and q_{hold} signals that an arbitrary finite amount of time can be waited without changing the accepted words ("the automaton is put on hold"). To get an intuition behind the enhancement consider the conjunction of a co-safety LTL formula and a fairness LTL formula and the equivalent automata. Since the co-safety part is decided on the finite prefixes, whereas the fairness part is decided on the infinite behavior, the two formula should be checked sequentially and not in parallel. For this, the fairness automaton is put on hold, i.e., his state is q_{hold} and the

co-safety automaton checks whether the co-safety part is satisfied. If a finite prefix has been consumed satisfying the co-safety part, the co-safety automaton switches to q_{acc} and the fairness automaton starts checking whether the fairness part is satisfied immediately.

DPLL-based positivity check. The succinct Emerson-Lei acceptance complicates checking positivity for MDPs (does "$\Pr^{\max}_{\mathcal{M}}(\Phi) > 0$ hold?"). In fact, [EL87] proved the NP-completeness of the analogous problem in automata theory: Does an automaton $\mathcal{A}$ with an Emerson-Lei acceptance Φ accept at least one word? The proof directly transfers to the positivity problem. We can extend this result in such a way, that for every class of Emerson-Lei acceptances, for which the satisfiability problem of the corresponding class of Boolean formulas is NP-complete, the positivity problem is NP-complete as well.

The strong connection between Boolean formulas and Emerson-Lei acceptance inspired us to a DPLL-based algorithm for checking positivity. As the DPLL-algorithm requires the input formula to be in conjunctive normal form, we modified Tseytin's transformation to provide an Emerson-Lei acceptance in conjunctive normal form. To achieve a polynomial size we introduce slack variables. In subsequent positivity we check for every MEC whether it contains an EC satisfying the acceptance condition. For this, we branch over the slack variables and the $\mathrm{Fin}(\cdot)$ variables. We keep track of a current EC, which may be pruned during the branching process in case of a $\mathrm{Fin}(\cdot)$ variable, and split up into several sub-ECs. By some heuristics, we can achieve a polynomial run time of our DPLL-based algorithm if the Emerson-Lei acceptances are Streett or generalized Rabin acceptance.

Experiments. For benchmarking our approach we consider the LTL-to-UBA translation and an integration of our DPLL-based positivity check into `PRISM`.

For our standard LTL benchmark set (with 94 formulas from [SB00; EH00; DAC99]) produced automata with a number of states comparable to `SPOT` and a slight advantage over `Rabinizer`. Overall, `Delag` produced automata with a minimal state space in 77 cases among `ltl2tgba` and `Rabinizer`, slightly surpassed by `ltl2tgba` with 78 formulas. `Delag` tends to produce automata with bigger acceptance condition than `SPOT`, but for those automata where `SPOT` produces a smaller acceptance condition, `Delag` produces equivalent automata with a smaller state space.

For evaluating the Emerson-Lei acceptance in the context of probabilistic model checking, we used the IEEE 802.11 WLAN handshaking protocol. We evaluated four LTL formulas on it, for two of them we asked for both the minimal and the maximal probability, obtaining overall seven properties. For each property, the running time of `PRISM` with `Delag` is comparable to the best running time among `PRISM` with `SPOT` and `PRISM` with `Rabinizer`, with one exception, where a heuristic applied to the automaton produced by `SPOT` but not to the automaton produced by `Delag`.

6 Conclusion

In this thesis we presented three new automata-based approaches different from the standard approach that employs deterministic Rabin automata in order to avoid or cope

with the double-exponential blow-up.

Good-for-games automata were a promising candidate for MDP analysis. We showed that a modified product construction and a standard MEC and reachability analysis are sufficient for MDP analysis against a GFG automaton specification. The overall approach starting from LTL can be done in double-exponential time, thus meeting the lower bound for MDP analysis under LTL specifications. Nonetheless, practical evaluation showed that good-for-games automata fell behind the standard approach of using [GO01] to generate a non-deterministic Büchi automaton and Safra's determinization [Saf88] to determinize it. The bad performance of the good-for-games approach in the benchmarks can be led back to the GFG automata generation algorithm developed by Henzinger and Piterman. This observation asks for an improved generation algorithm.

Unambiguous automata on the other hand showed an advantage over deterministic automata both on a theoretical level and in a practical evaluation. The single-exponential blow-up of the LTL-to-UBA translation gives UBA an edge over deterministic automata, but the actual model checking process becomes more complicated although still polynomial. Still, the performance of the overall process heavily depends on efficient LTL-to-UBA translators.

The principle that smaller automata likely lead to a more efficient model checking process also holds for our UBA-based approach to Markov chain analysis. The problem of UBA generation has not been studied in depth, but is current research by us. Currently, only one LTL-to-UBA translator is available, which uses an adaption of [Cou99], see also [Dur17]. A comparison in a symbolic setting as well as a comparison with the other single-exponential approaches of [BRV04] and [CSS03] is also still open.

The Emerson-Lei acceptance differs from unambiguity and good-for-games, as it works on the acceptance level, not on the non-determinism of the automaton. The product construction allows a reflection of the Boolean structure of the input LTL formula into the acceptance condition, not into the state space of the automaton. Since in our application to PMC, we rely on deterministic EL automata, the double-exponential blow-up from LTL to deterministic automata cannot be avoided. Still, in the automata generation part our tool `Delag` could compete with other standard automata tools like `Rabinizer` and `SPOT`.

In the application for model checking, EL automata are well-suited for the analysis of Markov chains, as checking positivity is still possible in polynomial time. For Markov decision processes, the situation becomes more complicated. The positivity problem relates closely to satisfiability of Boolean formulas, and thus, becomes NP-complete. As solution, we took inspiration from the DPLL algorithm. As an advantage, our DPLL-based algorithm works in polynomial time for the commonly used acceptances generalized Rabin and Streett. Our practical evaluation on the WLAN handshaking protocol shows, that it is worth to take the overhead of solving the NP-complete positivity problem, as the overhead is comparably small.

The translation of full LTL remains an open question. A reasonable starting point would be NBA and then a modification of a suitable determinization algorithm.

References

[AD94] Rajeev Alur and David L. Dill. "A Theory of Timed Automata". In: *Theoretical Computer Science* 126.2 (1994), pp. 183–235.

[Arn85] André Arnold. "Deterministic and non ambiguous rational omega-languages". In: *Automata on Infinite Words*. Vol. 192. Lecture Notes in Computer Science. Springer, 1985, pp. 18–27.

[Bab+15] Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. "The Hanoi Omega-Automata Format". In: *27th International Conference on Computer Aided Verification (CAV)*. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 479–486.

[Bai+16] Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, and James Worrell. "Markov Chains and Unambiguous Büchi Automata". In: *28th International Conference on Computer Aided Verification (CAV) - Part I*. Vol. 9779. Lecture Notes in Computer Science. Springer, 2016, pp. 23–42.

[Bel57] Richard Bellman. "A Markovian decision process". In: *Indiana Univ. Math. J.* 6 (1957), pp. 679–684.

[BGC09a] Christel Baier, Marcus Größer, and Frank Ciesinski. "Model Checking Linear-Time Properties of Probabilistic Systems". In: *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer, 2009. Chap. 13, pp. 519–570.

[BGC09b] Christel Baier, Marcus Größer, and Frank Ciesinski. "Quantitative Analysis under Fairness Constraints". In: *7th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 5799. Lecture Notes in Computer Science. Springer, 2009, pp. 135–150.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[BLW13] Michael Benedikt, Rastislav Lenhardt, and James Worrell. "LTL Model Checking of Interval Markov Chains". In: *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 7795. Lecture Notes in Computer Science. Springer, 2013, pp. 32–46.

[BLW14] Michael Benedikt, Rastislav Lenhardt, and James Worrell. *Model Checking Markov Chains Against Unambiguous Büchi Automata*. arXiv:1405.4560. 2014.

[BRV04] Doron Bustan, Sasha Rubin, and Moshe Y. Vardi. "Verifying ω-Regular Properties of Markov Chains". In: *16th International Conference on Computer Aided Verification (CAV)*. Vol. 3114. Lecture Notes in Computer Science. Springer, 2004, pp. 189–201.

[BY04] Johan Bengtsson and Wang Yi. “Timed Automata: Semantics, Algorithms and Tools”. In: *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. Vol. 3098. Lecture Notes in Computer Science. Springer, 2004, pp. 87–124.

[CES86] Edmund M. Clarke, Ernest A. Emerson, and A. Prasad Sistla. “Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications”. In: *ACM Transactions on Programming Languages and Systems* 8.2 (1986), pp. 244–263.

[Col09] Thomas Colcombet. “The Theory of Stabilisation Monoids and Regular Cost Functions”. In: *36 International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 5556. Lecture Notes in Computer Science. Springer, 2009, pp. 139–150.

[Col12] Thomas Colcombet. “Forms of Determinism for Automata”. In: *29th International Symposium on Theoretical Aspects of Computer Science, (STACS)*. Vol. 14. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 1–23.

[Col15] Thomas Colcombet. “Unambiguity in Automata Theory”. In: *17th International Workshop on Descriptional Complexity of Formal Systems (DCFS)*. Vol. 9118. Lecture Notes in Computer Science. Springer, 2015, pp. 3–18.

[Cou99] Jean-Michel Couvreur. “On-the-Fly Verification of Linear Temporal Logic”. In: *1st World Congress on Formal Methods in the Development of Computing Systems (FM)*. Vol. 1708. Lecture Notes in Computer Science. Springer, 1999, pp. 253–271.

[CSS03] Jean-Michel Couvreur, Nasser Saheb, and Grégoire Sutre. “An Optimal Automata Approach to LTL Model Checking of Probabilistic Systems”. In: *10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*. Vol. 2850. Lecture Notes in Computer Science. Springer, 2003, pp. 361–375.

[CY88] Costas Courcoubetis and Mihalis Yannakakis. “Verifying temporal properties of finite-state probabilistic programs”. In: *29th Symposium on Foundations of Computer Science (FOCS)*. IEEE computer society, 1988, pp. 338–345.

[CY95] Costas Courcoubetis and Mihalis Yannakakis. “The Complexity of Probabilistic Verification”. In: *Journal of the ACM* 42.4 (1995), pp. 857–907.

[DAC99] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. “Patterns in Property Specifications for Finite-State Verification”. In: *International Conference on Software Engineering (ICSE)*. ACM, 1999, pp. 411–420.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. “A Machine Program for Theorem-Proving”. In: *Communications of the ACM* 5.7 (1962), pp. 394–397.

[DP60] Martin Davis and Hilary Putnam. “A Computing Procedure for Quantification Theory”. In: *Journal of the ACM* 7.3 (1960), pp. 201–215.

[Dur14] Alexandre Duret-Lutz. “LTL translation improvements in Spot 1.0”. In: *International Journal of Critical Computer-Based Systems* 5.1/2 (2014), pp. 31–54.

[Dur17] Alexandre Duret-Lutz. “Contributions to LTL and ω-Automata for Model Checking”. Habilitation Thesis. Université Pierre et Marie Curie (Paris 6), 2017.

[EC80] Ernest A. Emerson and Edmund M. Clarke. “Characterizing Correctness Properties of Parallel Programs Using Fixpoints”. In: *7th International Conference on Automata, Languages and Programming (ICALP)*. Vol. 85. Lecture Notes in Computer Science. Springer, 1980, pp. 169–181.

[EH00] Kousha Etessami and Gerard J. Holzmann. “Optimizing Büchi Automata”. In: *11th International Conference on Concurrency Theory (CONCUR)*. Vol. 1877. Lecture Notes in Computer Science. Springer, 2000, pp. 153–167.

[EKS16] Javier Esparza, Jan Křetínský, and Salomon Sickert. “From LTL to Deterministic Automata - A Safraless Compositional Approach”. In: *Formal Methods in System Design* 49.3 (2016), pp. 219–271.

[EL87] Ernest A. Emerson and Chin-Laung Lei. “Modalities for Model Checking: Branching Time Logic Strikes Back”. In: *Science of Computer Programming* 8.3 (1987), pp. 275–306.

[FV96] Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes*. Springer, 1996.

[GO01] Paul Gastin and Denis Oddoux. “Fast LTL to Büchi Automata Translation”. In: *13th International Conference on Computer Aided Verification, (CAV)*. Vol. 2102. Lecture Notes in Computer Science. Springer, 2001, pp. 53–65.

[Hos04] Wolfgang Hoschek. *The Colt Distribution: Open Source Libraries for High Performance Scientific and Technical Computing in Java*. 2004.

[HP06] Thomas A. Henzinger and Nir Piterman. “Solving Games Without Determinization”. In: *20th International Workshop on Computer Science Logic (CSL)*. Vol. 4207. Lecture Notes in Computer Science. Springer, 2006, pp. 395–410.

[KB06] Joachim Klein and Christel Baier. “Experiments with deterministic ω-automata for formulas of linear temporal logic”. In: *Theoretical Computer Science* 363.2 (2006), pp. 182–195.

[Kle+14] Joachim Klein, David Müller, Christel Baier, and Sascha Klüppelholz. “Are Good-for-Games Automata Good for Probabilistic Model Checking?” In: *8th International Conference on Language and Automata Theory and Applications (LATA)*. Vol. 8370. Lecture Notes in Computer Science. Springer, 2014, pp. 453–465.

[KNP12] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. “The PRISM Benchmark Suite”. In: *9th International Conference on Quantitative Evaluation of SysTems (QEST)*. IEEE Computer Society, 2012, pp. 203–204.

[KNS02] Marta Zofia Kwiatkowska, Gethin Norman, and Jeremy Sproston. “Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol”. In: *Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification, (PAPM-PROBMIV)*. Vol. 2399. Lecture Notes in Computer Science. Springer, 2002, pp. 169–187.

[KPV06] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. “Safraless Compositional Synthesis”. In: *18th International Conference on Computer Aided Verification, (CAV)*. Vol. 4144. Lecture Notes in Computer Science. Springer, 2006, pp. 31–44.

[KR11] Orna Kupferman and Adin Rosenberg. “The Blow-up in Translating LTL to Deterministic Automata”. In: *6th International Workshop on Model Checking and Artificial Intelligence (MoChArt)*. Vol. 6572. Lecture Notes in Computer Science. Springer, 2011, pp. 85–94.

[KV05] Orna Kupferman and Moshe Y. Vardi. “From Linear Time to Branching Time”. In: *Transactions on Computational Logic* 6.2 (2005), pp. 273–294.

[KV15] Dileep Kini and Mahesh Viswanathan. “Limit Deterministic and Probabilistic Automata for $LTL \setminus \mathrm{G\ U}$”. In: *21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 9035. Lecture Notes in Computer Science. Springer, 2015, pp. 628–642.

[LP85] Orna Lichtenstein and Amir Pnueli. “Checking That Finite State Concurrent Programs Satisfy Their Linear Specification”. In: *12th Symposium on Principles of Programming Languages (POPL)*. ACM, 1985, pp. 97–107.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. “Model-Checking for Real-Time Systems”. In: *Fundamentals of Computation Theory*. Lecture Notes in Computer Science 965. 1995, pp. 62–88.

[LSS94] Nancy Lynch, Isaac Saias, and Roberto Segala. “Proving Time Bounds for Randomized Distributed Algorithms”. In: *13th Symposium on Principles of Distributed Computing (PODC)*. ACM, 1994, pp. 314–323.

[MS17] David Müller and Salomon Sickert. “LTL to Deterministic Emerson-Lei Automata”. In: *8th International Symposium on Games, Automata, Logics and Formal Verification (GandALF)*. Vol. 256. Electronic Proceedings of Theoretical Computer Science. Open Publishing Association, 2017, pp. 180–194.

[Pnu77] Amir Pnueli. “The Temporal Logic of Programs”. In: *18th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1977, pp. 46–57.

[PPS06] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. "Synthesis of Reactive(1) Designs". In: *7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Vol. 3855. Lecture Notes in Computer Science. Springer, 2006, pp. 364–380.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[Saf88] Shmuel Safra. "On The Complexity of ω-Automata". In: *29th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1988, pp. 319–327.

[SB00] Fabio Somenzi and Roderick Bloem. "Efficient Büchi Automata from LTL Formulae". In: *12th International Conference on Computer Aided Verification (CAV)*. Vol. 1855. Lecture Notes in Computer Science. Springer, 2000, pp. 248–263.

[SF07] Sven Schewe and Bernd Finkbeiner. "Bounded Synthesis". In: *Fifth Internation Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 4762. Lecture Notes in Computer Science. Springer, 2007, pp. 474–488.

[Sic+16] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. "Limit-Deterministic Büchi Automata for Linear Temporal Logic". In: *28th International Conference on Computer Aided Verification (CAV)*. Vol. 9780. Lecture Notes in Computer Science. Springer, 2016, pp. 312–332.

[Var85] Moshe Y. Vardi. "Automatic Verification of Probabilistic Concurrent Finite-State Programs". In: *26th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1985, pp. 327–338.

[Var99] Moshe Y. Vardi. "Probabilistic Linear-Time Model Checking: An Overview of the Automata-Theoretic Approach". In: *Fifth International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems (ARTS)*. Vol. 1601. Lecture Notes in Computer Science. Springer, 1999, pp. 265–276.

[VW86] Moshe Y. Vardi and Pierre Wolper. "An Automata-Theoretic Approach to Automatic Program Verification". In: *1st Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 1986, pp. 332–344.