LTL to Deterministic Emerson-Lei Automata

David Müller
Technische Universität Dresden *
david.mueller2@tu-dresden.de

Salomon Sickert
Technische Universität München †
sickert@in.tum.de

We introduce a new translation from linear temporal logic (LTL) to deterministic Emerson-Lei automata, which are ω-automata with a Muller acceptance condition symbolically expressed as a Boolean formula. The richer acceptance condition structure allows the shift of complexity from the state space to the acceptance condition. Conceptually the construction is an enhanced product construction that exploits knowledge of its components to reduce the number of states. We identify two fragments of LTL, for which one can easily construct deterministic automata and show how knowledge of these components can reduce the number of states. We extend this idea to a general LTL framework, where we can use arbitrary LTL to deterministic automata translators for parts of formulas outside the mentioned fragments. Further, we show succinctness of the translation compared to existing construction. The construction is implemented in the tool `Delag`, which we evaluate on several benchmarks of LTL formulas and probabilistic model checking case studies.

1 Introduction

Deterministic ω-automata play an essential role in the verification of probabilistic systems and in the synthesis of reactive systems, which generally prohibit a direct use of non-deterministic automata. However, determinisation of non-deterministic automata may cause an exponential blow-up, which makes these applications computationally hard. Hence there exists a long line of research aiming at shrinking the size of the generated deterministic automata as far as possible. All these translations have in common that they target a specific acceptance condition, such as Rabin, Streett, or Parity, and thus have to sometimes store progress information of the acceptance condition in the state.

In this work, we reexamine the Muller acceptance condition with a crucial twist: Instead of an explicit representation, we represent the acceptance condition in a symbolic fashion, as presented in [2], which we call Emerson-Lei acceptance. Moving to a compactly expressed acceptance condition allows us to reduce the number of states and to use fewer acceptance sets compared to existing translations, although there is a well-known exponential lower bound for the size of deterministic ω-automata starting from a non-deterministic ω-automaton [25]. Of course algorithms need to be adapted to this more complex scenario, but we present examples where this reduces the time needed for probabilistic model checking.

Related Work. There are two lines of research to cope with the exponential blow-up caused by determinisation of ω-automata. The first explores restricted forms of non-determinism that are still usable for probabilistic verification, such as limit-deterministic automata [31, 5, 28, 29] or good-for-games-automata [14, 18] for Markov decision processes, or unambiguous Büchi automata for Markov chains [4]. The authors of [13] try to avoid the full Safra's determinisation by under-approximating and over-approximating it via break-point and powerset construction. In the context of synthesis, one can evade determinisation using universal co-Büchi tree automata instead of deterministic parity automata [19].

*This work is funded by the DFG-project BA-1679/12-1 and partially funded by the DFG Research Training Group "QuantLA: Quantitative Logics and Automata" (GRK 1763)

†This work is funded by the DFG Research Training Group "PUMA: Programm- und Modell-Analyse" (GRK 1480)

To appear in EPTCS.

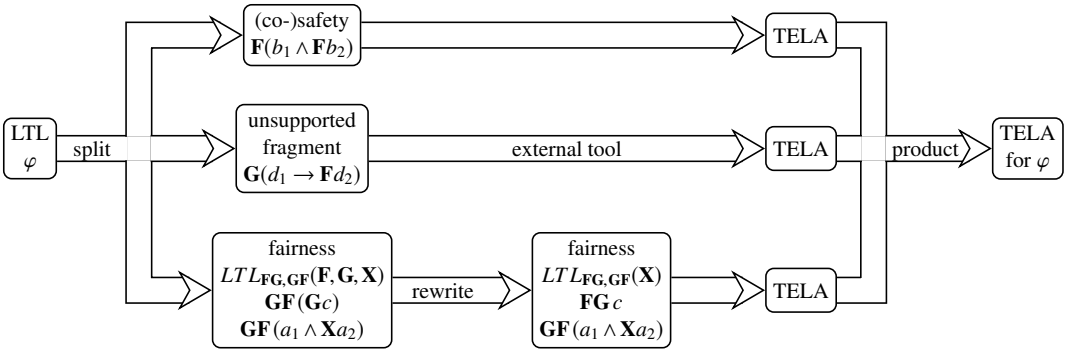


Figure 1: The input LTL formula is split up, each subformula is translated independently, and then a product automaton is constructed, as can be seen for the example $\varphi = \mathbf{GF}(a_1 \wedge \mathbf{X}a_2) \wedge \mathbf{F}(b_1 \wedge \mathbf{F}b_2) \wedge \mathbf{GF}(\mathbf{G}c) \wedge \mathbf{G}(c_1 \rightarrow \mathbf{F}c_2)$.

The second line of research aims at reducing the size of the state space of the resulting deterministic automaton. The most prominent determinisation method, Safra's determinisation, translates a non-deterministic Büchi into a deterministic Rabin automaton [26]. This translation is implemented in `ltl2dstar` with several heuristics [16, 17]. In the last decades there has been a lot of progress on determinisation of Büchi automata refining Safra's construction [23, 15, 21, 27, 24, 11]. While there still remains the exponential lower bound, efficient implementations are also available in `SPOT` [6]. There has been also work on direct translations starting with fragments or even full LTL, see the history of `Rabinizer` [3, 9]. The approach of [28] originates from the same family of translations, which together with [8], yields an asymptotically optimal translation from LTL (via limit-deterministic automata) to Parity automata, which is implemented in `ltl2dpa`. The authors of [22] follow a compositional approach where the LTL formula is brought into a normal form, decomposed, and then subformulas are translated separately. However, all these constructions target a specific acceptance condition structure — Rabin, Streett, or Parity — and thus sometimes need to encode the progress of the acceptance condition in the state space of the resulting automaton.

Contribution. We present a translation from LTL to deterministic Emerson-Lei automata that trades a compact state space for a more complex acceptance condition structure. There has been previously the idea of a product construction relying on known translations in [2] to obtain a more complex acceptance condition. Here, we give a direct translation of fragments of LTL without an intermediate step over non-deterministic automata. We consider special liveness properties in particular and give a translation based on buffers. For safety and cosafety LTL formulas we rely on the *af* function [9, 28] 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. A general scheme for our approach is depicted in Figure 1, which we implemented in the tool `Delag` (Deterministic Emerson-Lei Automata Generator).

We conducted several experiments to evaluate the practical impact of this idea: At first we compared the size of the automata measured in state space size as well as acceptance sizes for our tool and several other tools like `SPOT` and `Rabinizer`. Secondly, we performed a case study (IEEE 802.11 Wireless LAN Handshaking protocol) and also compared it with `SPOT` and `Rabinizer`. On both sides, we could show the potential of `Delag`, i.e., allowing arbitrary acceptance conditions to obtain smaller automata. The implementation and additional material can be found at [1].

2 Preliminaries

2.1 Linear Temporal Logic

We consider standard linear temporal logic (LTL) with all negations pushed down to the propositions.

Definition 1 (LTL)**.** *A formula of LTL in* negation normal form *over a finite set of atomic propositions* (Ap) *is given by the syntax:*

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid \varphi \mathbf{R} \varphi \qquad \text{with } a \in Ap$$

Furthermore, we introduce the abbreviations: $\mathbf{F}\varphi := \mathbf{tt}\mathbf{U}\varphi$, $\mathbf{G}\varphi := \mathbf{ff}\mathbf{R}\varphi$. *An ω-word w is an infinite sequence of sets of atomic propositions $w[0]w[1]w[2]\cdots$ and we denote the infinite suffix $w[i]w[i+1]\cdots$ by w_i. The satisfaction relation $\models$ between ω-words and formulas is inductively defined as follows:*

$w \models \mathbf{tt}$		$w \not\models \mathbf{ff}$	$w \models \mathbf{X}\varphi$	*iff*	$w_1 \models \varphi$
$w \models a$	*iff*	$a \in w[0]$	$w \models \varphi \mathbf{U} \psi$	*iff*	$\exists i.\, w_i \models \psi$ *and* $\forall j < i. w_j \models \varphi$
$w \models \neg a$	*iff*	$a \notin w[0]$	$w \models \varphi \mathbf{R} \psi$	*iff*	$\forall i.\, w_i \models \psi$ *or*
$w \models \varphi \wedge \psi$	*iff*	$w \models \varphi$ *and* $w \models \psi$			$\exists i.\, w_i \models \varphi$ *and* $\forall j \leq i. w_j \models \psi$
$w \models \varphi \vee \psi$	*iff*	$w \models \varphi$ *or* $w \models \psi$			

Two formulas φ, ψ are called equivalent, denoted $\varphi \equiv \psi$, if $w \models \varphi \leftrightarrow w \models \psi$ for all $w \in (2^{Ap})^\omega$. $\mathsf{sf}(\varphi)$ *is defined as the set of temporal subformulas (*$\mathbf{U}, \mathbf{R}, \mathbf{X}$*) not nested within the scope of another temporal operator, e.g.,* $\mathsf{sf}((\mathbf{FG}a) \vee \mathbf{X}b) = \{\mathbf{FG}a, \mathbf{X}b\}$.

2.2 Fragments of LTL

We study several syntactic fragments of LTL. Let us denote by $LTL(\mathcal{X})$ the syntactic restriction of LTL to the temporal operators of $\mathcal{X}$. Furthermore we allow to denote prefixes that are applied to all formulas by a subscript: $LTL_{X,Y}(\mathcal{X}) = \{X\varphi, Y\varphi \mid \varphi \in LTL(\mathcal{X})\}$. We now identify three (well-known) syntactic LTL fragments commonly used in system property specifications:

- safety: $LTL(\mathbf{R}, \mathbf{X})$
- reachability (or cosafety): $LTL(\mathbf{U}, \mathbf{X})$
- fairness: $LTL_{\mathbf{FG},\mathbf{GF}}(\mathbf{F}, \mathbf{G}, \mathbf{X})$

We now show that the last fragment can be simplified to formulas without nested **F** and **G**:

Theorem 1 (Fairness LTL Normal Form)**.** *Let φ be an $LTL_{\mathbf{FG},\mathbf{GF}}(\mathbf{F}, \mathbf{G}, \mathbf{X})$ formula. Then there exists an equivalent formula $\varphi' \equiv \varphi$ that is a boolean combination of formulas in $LTL_{\mathbf{FG},\mathbf{GF}}(\mathbf{X})$.*

Proof. Exhaustive application of the following folklore equivalence-preserving rewrite rules, described in [10, 30, 20], brings every fairness LTL formula into the desired normal form:

$\mathbf{FG}(\mathbf{F}\varphi)$	$\mapsto$	$\mathbf{GF}\varphi$	$\mathbf{GF}(\mathbf{F}\varphi)$	$\mapsto$	$\mathbf{GF}\varphi$
$\mathbf{FG}(\mathbf{G}\varphi)$	$\mapsto$	$\mathbf{FG}\varphi$	$\mathbf{GF}(\mathbf{G}\varphi)$	$\mapsto$	$\mathbf{FG}\varphi$
$\mathbf{FG}(\mathbf{X}\varphi)$	$\mapsto$	$\mathbf{FG}\varphi$	$\mathbf{GF}(\mathbf{X}\varphi)$	$\mapsto$	$\mathbf{GF}\varphi$
$\mathbf{FG}(\varphi \wedge \psi)$	$\mapsto$	$\mathbf{FG}\varphi \wedge \mathbf{FG}\psi$	$\mathbf{GF}(\varphi \vee \psi)$	$\mapsto$	$\mathbf{GF}\varphi \vee \mathbf{GF}\psi$
$\mathbf{FG}(\varphi \vee \mathbf{F}\psi)$	$\mapsto$	$\mathbf{FG}\varphi \vee \mathbf{GF}\psi$	$\mathbf{GF}(\varphi \wedge \mathbf{F}\psi)$	$\mapsto$	$\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$
$\mathbf{FG}(\varphi \vee \mathbf{G}\psi)$	$\mapsto$	$\mathbf{FG}\varphi \vee \mathbf{FG}\psi$	$\mathbf{GF}(\varphi \wedge \mathbf{G}\psi)$	$\mapsto$	$\mathbf{GF}\varphi \wedge \mathbf{FG}\psi$
$\varphi \notin LTL(\mathbf{X}) \Rightarrow \mathbf{FG}(\varphi)$	$\mapsto$	$\mathbf{FG}(\mathsf{cnf}(\varphi))$	$\varphi \notin LTL(\mathbf{X}) \Rightarrow \mathbf{GF}(\varphi)$	$\mapsto$	$\mathbf{GF}(\mathsf{dnf}(\varphi))$

with $\mathsf{cnf}(\varphi)$ and $\mathsf{dnf}(\varphi)$ denoting the translation into conjunctive and disjunctive normal form. □

This translation might cause an exponential blow-up in formula size due to the translation into conjunctive and disjunctive normal form. However, the construction for fairness LTL to deterministic automata we present is only dependent on the size of the alphabet and the nesting depth of the **X**-operators, which are both unchanged (or even decreased) by the translation. Further from now on we assume all fairness LTL formulas are rewritten to this normal form.

Apart from the rules listed above, our implementation uses several well-known simplification rules to rewrite formulas outside of the fairness fragment to formulas within, e.g., $\mathbf{GF}(\varphi\mathbf{U}\psi) \mapsto \mathbf{GF}\psi$ and $\mathbf{FG}(\varphi\mathbf{U}\psi) \mapsto \mathbf{GF}\psi \wedge \mathbf{FG}(\varphi \vee \psi)$.

2.3 Deterministic Emerson-Lei Automata

Emerson-Lei automata are Muller automata with their acceptance condition expressed as a *generic acceptance condition* (see [2]): Instead of representing every Muller set explicitly a symbolic representation is used. We will take as acceptance condition a Boolean combination over the atomic propositions $Fin(P)$ and $Inf(P)$ where P is an arbitrary subset of transitions of an ω-automaton $\mathcal{A}$. We denote the set of all generic acceptance conditions by C_δ.

Definition 2 (Deterministic Transition-Based Emerson-Lei Automata)**.** *A deterministic transition-based Emerson-Lei automaton (TELA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \alpha)$ where Q is a finite set of states, Σ is an alphabet, $\delta\colon Q \times \Sigma \to Q$ is a transition function, q_0 is the initial state, and $\alpha \in C_\delta$ is a generic acceptance condition. Furthermore we use a superscript to denote a component of a specific automaton, e.g., $\delta^{\mathcal{A}}$ is the transition function of $\mathcal{A}$.*

For convenience we sometimes interpret the transition function as a relation and write $(q, a, q') \in \delta$ instead of $q' = \delta(q, a)$. A run ρ of a TELA $\mathcal{A}$ on the ω-word w is an infinite sequence of transitions $\rho = (q_0, w[0], q_1)(q_1, w[1], q_2)\cdots$ respecting the transition function, i.e. $\rho[i] = (q_i, w[i], q_{i+1}) \in \delta$ for every $i \geq 0$. We denote by $\inf(\rho)$ the set of transitions occurring infinitely often in the run. A run is called accepting for $Fin(P)$ if $\inf(\rho) \cap P = \emptyset$ and accepting for $Inf(P)$ if $\inf(\rho) \cap P \neq \emptyset$. For arbitrary acceptance conditions φ, i.e., Boolean combinations of $Inf(P)$ and $Fin(P)$, a run is accepting if $\inf(\rho)$ satisfies φ in the expected way. All well-known acceptance conditions, such as Büchi, Rabin, Streett, and Parity, can be expressed easily using this mechanism.

Since $Inf(P)$ and $Fin(P)$ are dual, one can complement a deterministic TELA just by complementing the acceptance condition, i.e., replacing every occurrence of $Inf(P)$ with $Fin(P)$, every occurrence of $Fin(P)$ with $Inf(P)$, and every disjunction with conjunction and every conjunction with a disjunction.

3 Construction

The automaton is constructed from an LTL formula as a product of smaller automata for each temporal subformula. We identified several fragments of LTL in the preliminaries and now present specialised constructions for each of them. While the standard product construction yields an automaton in the size of the product of all automata in the worst-case, the structure of the formula enables us to propagate information, such that we can suspend or disable automata of the product depending on the context.

Consider the following parametric formula: $\mathbf{GF}(a_1 \wedge \mathbf{X}(a_2 \wedge \ldots \mathbf{X}a_m)) \wedge \mathbf{F}(b_1 \wedge \mathbf{F}(b_2 \wedge \ldots \mathbf{F}b_n))$. We will later demonstrate that the propagation of information allows us to construct a Büchi automaton of size $O(n+m)$, while `SPOT` in the standard configuration yields automata of size $O(n \cdot m)$ and only after enabling simulation-based reductions this decreases to sizes comparable to our automata. Let us now examine the construction, while we translate the formula $\mathbf{GF}(a_1 \wedge \mathbf{X}a_2) \wedge \mathbf{F}(b_1 \wedge \mathbf{F}b_2)$.

3.1 Fairness-LTL

First, we consider the fairness fragment and show that there is a natural way to represent it as deterministic automata. In particular, if we look at Boolean combinations of fairness-LTL formulas ($LTL_{\mathbf{FG},\mathbf{GF}}(\mathbf{X})$), we obtain an acceptance condition mirroring the structure of the input formula. Furthermore, if the formula does not contain any $\mathbf{X}$, the automata we obtain is a single-state automaton. For all other formulas we need to store a bounded history in the form of a FIFO-buffer of seen sets of atomic propositions (or valuations). We will now establish the tools necessary to compute the structure of such a buffer. We use the following operations defined on finite and infinite sequences of sets (assuming $n \leq m$):

$$
\begin{array}{rlcl}
\text{Pointwise Intersection:} & u[0]u[1]\ldots \sqcap v[0]\ldots v[m] & = & (u[0]\cap v[0])\ldots(u[m]\cap v[m])\emptyset^\omega \\
\text{Pointwise Union:} & u[0]\ldots u[n] \sqcup v[0]\ldots v[m] & = & (u[0]\cup v[0])\ldots(u[n]\cup v[n])\ldots v[m] \\
\text{Forward Closure:} & \mathsf{cl}(w[0]\ldots w[n]) & = & w[0](w[0]\cup w[1])\ldots\bigcup_{k=0}^{n} w[k] \\
\text{Drop Last Set of Letters:} & \mathsf{drop}(w[0]\ldots w[n]w[n+1]) & = & w[0]\ldots w[n]
\end{array}
$$

Relevant History. Let us consider our example formula: $\mathbf{GF}(a_1 \wedge \mathbf{X}a_2)$. In order to check whether $w \models a_1 \wedge \mathbf{X}a_2$ holds we just need to know whether $a_1 \in w[0]$ and $a_2 \in w[1]$ holds. The rest of the w can be projected away. The *relevant history* $\mathcal{H}(\varphi)$ for an LTL formula φ is a finite word over 2^{AP} and masks all propositions that are irrelevant for evaluating φ. We compute the relevant history $\mathcal{H}$ recursively from the structure of the formula:

$$\mathcal{H}\colon LTL(\mathbf{X}) \to (2^{Ap})^*$$

$$
\begin{array}{rclcrcll}
\mathcal{H}(\mathbf{tt}) & = & \epsilon & & \mathcal{H}(\mathbf{ff}) & = & \epsilon & \mathcal{H}(\mathbf{X}\varphi) = \emptyset\mathcal{H}(\varphi) \\
\mathcal{H}(a) & = & \{a\} & & \mathcal{H}(\neg a) & = & \{a\} & \\
\mathcal{H}(\varphi\wedge\psi) & = & \mathcal{H}(\varphi) \sqcup \mathcal{H}(\psi) & & \mathcal{H}(\varphi\vee\psi) & = & \mathcal{H}(\varphi) \sqcup \mathcal{H}(\psi) &
\end{array}
$$

Lemma 1. *Let φ be an $LTL(\mathbf{X})$ formula and let w be a ω-word. Then $w \models \varphi$ if and only if $w \sqcap \mathcal{H}(\varphi) \models \varphi$.*

Proof. By induction on φ. For succinctness we just exhibit two cases and all other cases are analogous.
Case $\varphi = \mathbf{X}\psi$. Then $w \models \varphi$ iff $w_1 \models \psi$ iff $w_1 \sqcap \mathcal{H}(\psi) \models \psi$ iff $\emptyset(w_1 \sqcap \mathcal{H}(\psi)) \models \varphi$ iff $w \sqcap \mathcal{H}(\varphi) \models \varphi$.
Case $\varphi = \psi \wedge \psi'$. Then $w \models \varphi$ iff $w \models \psi \wedge w \models \psi'$ iff $w \sqcap \mathcal{H}(\psi) \models \psi \wedge w \sqcap \mathcal{H}(\psi') \models \psi'$ iff $w \sqcap (\mathcal{H}(\psi) \sqcup \mathcal{H}(\psi')) \models \varphi$ iff $w \sqcap \mathcal{H}(\varphi) \models \varphi$. □

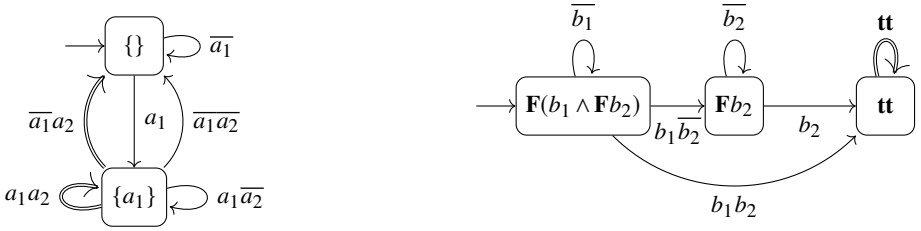
The TELA we are constructing keeps a buffer masked by $\mathcal{H}$. Intuitively the automaton delays the decision whether φ holds by $n = |\mathcal{H}(\varphi)| - 1$ steps and then decides whether it holds true, instead of non-deterministically guessing the future and verifying this guess as done in standard LTL translations.

Definition 3. *Let φ be an $LTL(\mathbf{X})$ formula over Ap and let $n = \max(|\mathcal{H}(\varphi)| - 1, 0)$. We then define one TELA for $\mathbf{GF}\varphi$:*

$$\mathcal{A}(\mathbf{GF}\varphi) = (Q, 2^{Ap}, \delta, \emptyset^n, Inf(\alpha))$$

$$
\begin{array}{lcl}
Q & = & \{w \in (2^{Ap})^n \mid \forall i.\ w[i] \subseteq \mathsf{cl}(\mathcal{H}(\varphi))[i]\} \\
\delta(\nu w, \nu') & = & w\nu' \sqcap \mathsf{drop}(\mathsf{cl}(\mathcal{H}(\varphi))) \quad \textit{for all } \nu, \nu' \in 2^{Ap} \textit{ and } w \in (2^{Ap})^{n-1} \\
\alpha & = & \{(w, \nu, w') \in \delta \mid w\nu\emptyset^\omega \models \varphi\}
\end{array}
$$

Observe that we must take the closure of $\mathcal{H}(\varphi)$ before intersecting with the buffer. Otherwise we might lose information while propagating letters from the back to the front of the buffer. Further, we can always drop the last set of letters of the relevant history, since a transition-based acceptance is used. In the context of state-based acceptance this needs to be also stored in the buffer.



(a) $\psi_1 = \mathbf{GF}(a_1 \wedge \mathbf{X}a_2)$ (b) $\psi_2 = \mathbf{F}(b_1 \wedge \mathbf{F}b_2)$

Figure 2: Automata for ψ_1 and ψ_2. Bold edges denote accepting transitions.

Let us apply this construction to our example: $\mathbf{GF}(a_1 \wedge \mathbf{X}a_2)$. First, we get $\mathcal{H}(a_1 \wedge \mathbf{X}a_2) = \{a_1\}\{a_2\}$. Second, since we always drop the last set of letters, we have $\mathsf{drop}(\mathcal{H}(a_1 \wedge \mathbf{X}a_2)) = \{a_1\}$ and $n = 1$. Thus we obtain the TELA automaton shown in Figure 2a, which is in fact a Büchi automaton.

Theorem 2. *Let φ be an $LTL(\mathbf{X})$ formula over Ap.*

$$L(\mathbf{GF}\varphi) = L(\mathcal{A}(\mathbf{GF}\varphi))$$

Proof. Assume $w \models \mathbf{GF}\varphi$ holds. Thus we have $\exists^\infty i.\ w_i \models \varphi$ and we obtain $\exists^\infty i.\ w_i \sqcap \mathcal{H}(\varphi) \models \varphi$ by using Lemma 1. Thus there exists a finite word $w' \in 2^{Ap}$ with (1) $w'\emptyset^\omega = w_i \sqcap \mathcal{H}(\varphi)$, (2) $w'\emptyset^\omega \models \varphi$, and (3) $|w'| = |\mathcal{H}(\varphi)|$. Thus $\mathcal{A}(\mathbf{GF}\varphi)$ infinitely often takes the (shortened) transition $t = (w'[0]\dots w'[n-1], w'[n])$. Due to (2) we have $t \in \alpha$ and thus $w \in L(\mathcal{A}(\mathbf{GF}\varphi))$. The other direction is analogous. □

Since $\mathbf{FG}$ is equivalent to $\neg\mathbf{GF}\neg\varphi$, we immediately obtain also a translation for $LTL_{\mathbf{FG}}(\mathbf{X})$. We only need to change the acceptance condition to $Fin(\alpha)$ with $\alpha = \{(w, v, w') \in \delta \mid wv\emptyset^\omega \not\models \varphi\}$.

3.2 Safety- and Cosafety-LTL

Translating safety LTL to deterministic automata is a well-studied problem. Since these languages can be defined using bad prefixes, meaning once a bad prefix has been read, the word is rejected, most automata generated by most available translations will have a single rejecting sink. All other states and transitions are then either rejecting or accepting. We use the straight-forward approach to apply the *af*-function from [9] to obtain a deterministic automaton for cosafety LTL formulas and by duality also for automata for safety languages. The *af*-function computes the left-derivative of a language expressed as an LTL formula.

Definition 4 ([9], Definition 7)**.** *Let φ be a formula of $LTL(\mathbf{U}, \mathbf{X})$, then*

$$\mathcal{A}(\varphi) = (Q, 2^{Ap}, \delta, [\varphi]_P, \{[\mathbf{tt}]_P\}).$$

Theorem 3 ([9], Theorem 2)**.** *Let φ be a formula of $LTL(\mathbf{U}, \mathbf{X})$, then*

$$L(\varphi) = L(\mathcal{A}(\varphi)).$$

For the cosafety formula $\mathbf{F}(b_1 \wedge \mathbf{F}b_2)$ we then obtain the automaton of Figure 2b with the accepting sink $[\mathbf{tt}]_P$. This approach also immediately tells us, when a run is accepting by looking at the state.

3.3 General LTL

If the translation encounters a subformula not covered by Section 3.1 and Section 3.2, it resorts to an external general purpose LTL to deterministic automaton translation. Here no restrictions on the type of the automaton are made, since all of them — Rabin, Streett, Parity, Büchi — can be interpreted as a TELA.

3.4 Product construction

Standard Construction. All these deterministic automata are then combined using a product construction. We first introduce the standard product construction for Emerson-Lei Automata that is similar to the product construction for Muller automata and then move on to the enhanced construction.

Definition 5. *Let φ be a formula and for every $\psi \in \mathsf{sf}(\varphi)$ let $\mathcal{A}(\psi)$ be a deterministic TELA recognising $L(\psi)$. The deterministic TELA for the product automaton is defined as:*

$$\mathcal{A}^{\times}(\varphi) = (Q, 2^{Ap}, \delta, q_0, \alpha(\varphi))$$

$$\delta(s, \nu) = \{\psi \mapsto \delta^{\mathcal{A}(\psi)}(s[\psi], \nu) \mid \psi \in \mathsf{sf}(\varphi)\} \qquad q_0 = \{\psi \mapsto q_0^{\mathcal{A}(\psi)} \mid \psi \in \mathsf{sf}(\varphi)\}$$

We denote by $s[\psi] = q$ the current state of the automaton $\mathcal{A}(\psi)$ in the product state s, meaning $\psi \mapsto q \in s$. Since all $\delta^{\mathcal{A}}(\psi)$ are deterministic, δ is also deterministic. We denote by $Q^{\mathcal{A}(\psi)}$ the states of $\mathcal{A}(\psi)$ and by $q_0^{\mathcal{A}(\psi)}$ the initial state of $\mathcal{A}(\psi)$. Further Q is defined as the set of all from the initial state reachable states. The acceptance condition is recursively computed over the structure of φ with $\uparrow$ denoting the lifting of the acceptance condition:

$$\begin{array}{lcllcl} \alpha(\mathbf{tt}) & = & \mathbf{tt} & \alpha(\varphi \wedge \psi) & = & \alpha(\varphi) \wedge \alpha(\psi) \\ \alpha(\mathbf{ff}) & = & \mathbf{ff} & \alpha(\varphi \vee \psi) & = & \alpha(\varphi) \vee \alpha(\psi) \end{array} \qquad \alpha(\psi) \;=\; \uparrow \alpha^{\mathcal{A}(\psi)}$$

Theorem 4. *Let φ be an LTL formula. Then*

$$L(\varphi) = L(\mathcal{A}^{\times}(\varphi))$$

Enhanced Construction. An essential part of the enhanced product construction is the removal of unnecessary information from the product states. For this we introduce three additional states with special semantics: q_{acc} signalises that the component moved to an accepting trap, while q_{rej} expresses that the component moved to a rejecting trap. Alternatively, if a component got irrelevant for the acceptance condition it is also moved to q_{rej}. Lastly, q_{hold} says that the component was put on hold. More specifically, we put the fairness automata on hold, if a "neighbouring" automaton still needs to fulfil its goal, such as reaching an accepting trap. To make notation easier to read we assume that every automaton $\mathcal{A}(\varphi)$ contains these states and all accepting sinks (or traps) have been replaced by q_{acc} and rejecting by q_{rej}. In the following we use the following abbreviations to reason about LTL formulas:

- $\mathsf{conj}(\varphi)$ ($\mathsf{disj}(\varphi)$) denotes the set of all conjuncts of a conjunction (disjuncts of a disjunction) outside the scope of a temporal operator, e.g. let $\varphi = \mathbf{F}a \wedge (\mathbf{X}b \vee \mathbf{G}c)$, then $\mathsf{conj}(\varphi) = \{\{\mathbf{F}a, \mathbf{X}b \vee \mathbf{G}c\}\}$ and $\mathsf{disj}(\varphi) = \{\{\mathbf{X}b, \mathbf{G}c\}\}$.
- $\varphi[\Psi/\psi]$ denotes the substitution of all formulas in the set Ψ with the formula ψ, e.g. $(\mathbf{F}a \wedge (\mathbf{X}b \vee \mathbf{G}c))[\{\mathbf{F}a, \mathbf{G}a\}/\mathbf{tt}] = \mathbf{tt} \wedge (\mathbf{X}b \vee \mathbf{G}c)$.

- $\mathsf{support}(\varphi)$ denotes the support of a formula, where the formula is viewed as a propositional formula, which means that temporal operators are also considered propositions, e.g. $\mathsf{support}((\mathbf{X}a \wedge \mathbf{F}b) \vee (\mathbf{F}b)) = \{\mathbf{F}b\}$. This means every assignment can be restricted to the propositions of the support: $S \models_P \varphi \leftrightarrow S \cap \mathsf{support}(\varphi) \models_P \varphi$, where $\models_P$ denotes the conventional propositional satisfaction relation.

We use the following definitions to manipulate product states:

Definition 6 (Product State Modifications)**.** *An* update *of a product state tests a predicate P on a formula-state pair (ψ, q) and replaces q with a new value obtained by the updater U depending on ψ, if it holds:*

$$\mathsf{update}(s, P, U) = \{\psi \mapsto (\textbf{\textit{if}}\ P(\psi, q)\ \textbf{\textit{then}}\ U(\psi)\ \textbf{\textit{else}}\ q) \mid \psi \mapsto q \in s\}$$

$\mathsf{prune}(s)$ *disables automata in s that became irrelevant for the acceptance condition, meaning there are no longer in the support of the original formula after using knowledge from other automata. For this let us denote by Ψ_{acc} all $\psi \mapsto q_{\mathsf{acc}} \in s$ and by Ψ_{rej} all $\psi \mapsto q_{\mathsf{rej}} \in s$.*

$$\begin{array}{ll} \mathsf{prune}(s) & = \mathsf{update}(s, P, U) \\ P(\psi, q) & = (q \neq q_{\mathsf{acc}} \wedge \psi \notin \mathsf{support}(\varphi[\Psi_{\mathsf{acc}}/\mathbf{tt}, \Psi_{\mathsf{rej}}/\mathbf{ff}])) \\ U(\psi) & = q_{\mathsf{rej}} \end{array}$$

$\mathsf{run}(s)$ *starts (fairness) automata that are required for the acceptance but have been put on hold. This is the case, if automata with terminal acceptance for formulas in the same conjunction ($\mathsf{run}(s)_c$) have not yet reached q_{acc} or the dual case for disjunctions:*

$$\begin{array}{ll} \mathsf{run}(s)_c & = \mathsf{update}(s, P_c, U) \\ \mathsf{run}(s)_d & = \mathsf{update}(s, P_d, U) \\ P_c(\psi, q) & = (q = q_{\mathsf{hold}} \wedge \exists C \in \mathsf{conj}(\varphi).\ \psi \in C \wedge \forall \chi \in C \cap LTL(\mathbf{U}, \mathbf{X}).\ s[\chi] = q_{\mathsf{acc}}) \\ P_d(\psi, q) & = (q = q_{\mathsf{hold}} \wedge \exists D \in \mathsf{disj}(\varphi).\ \psi \in D \wedge \forall \chi \in D \cap LTL(\mathbf{R}, \mathbf{X}).\ s[\chi] = q_{\mathsf{rej}}) \\ U(\psi) & = q_0^{\mathcal{A}(\psi)} \end{array}$$

Definition 7 (Enhanced Product Automaton)**.** *Let φ be a formula. The TELA for the enhanced product automaton is defined the same way as Definition 5 with the following changes:*

$$\mathcal{A}_E^{\times}(\varphi) = (Q, 2^{Ap}, \delta, q_0, \alpha(\varphi))$$

$$\begin{array}{rcl} \delta(s, \nu) & = & \mathsf{run}\left(\mathsf{prune}(\{\psi \mapsto \delta^{\mathcal{A}(\psi)}(s[\psi], \nu) \mid \psi \in \mathsf{sf}(\varphi)\})\right) \\ q_0 & = & \mathsf{run}\left(\left\{\psi \mapsto \begin{cases} q_0^{\mathcal{A}(\psi)} & \text{if } \psi \in \mathsf{sf}(\varphi) \setminus LTL_{\mathbf{FG},\mathbf{GF}}(\mathbf{X}) \\ q_{\mathsf{hold}} & \text{otherwise} \end{cases}\right\}\right) \end{array}$$

Theorem 5. *Let φ be a formula.*

$$L(\varphi) = L(\mathcal{A}_E^{\times}(\varphi))$$

If we apply this construction to $\mathbf{GF}(a_1 \wedge \mathbf{X}a_2) \wedge \mathbf{F}(b_1 \wedge \mathbf{F}b_2)$ we obtain the automaton shown in Figure 3. Observe that ψ_2 is put on hold until the automaton for ψ_1 reaches q_{acc}.

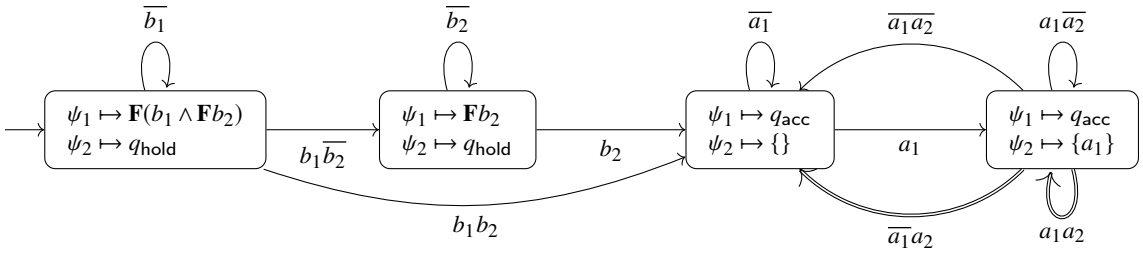


Figure 3: Enhanced Product Automaton for $\mathbf{GF}(a_1 \wedge \mathbf{X}a_2) \wedge \mathbf{F}(b_1 \wedge \mathbf{F}b_2)$, only the accepting edges for ψ_2 are drawn.

3.4.1 Further Optimisations

There are two further optimisations we implement: First, we replace the local histories of each automaton for $LTL_{\mathbf{FG},\mathbf{GF}}(\mathbf{X})$ with one *global* history. Second, we *piggyback* the acceptance of (co-)safety automata on neighbouring fairness automata. Let $C \in \mathsf{conj}(\varphi)$ be a conjunction, $\psi_r \in LTL(\mathbf{U},\mathbf{X}) \cap C$ and $\psi_f \in LTL_{\mathbf{FG}}(\mathbf{X}) \cap C$. We then have $\alpha^{\mathcal{A}(\psi_f)} = Fin(S)$ and extend S with $Q^{\mathcal{A}(\psi_r)} \setminus \{q_{\mathsf{acc}}\}$. The same trick can be applied to $\psi_f \in LTL_{\mathbf{GF}}(\mathbf{X})$ and of course to the dual case with $\psi_s \in LTL(\mathbf{R},\mathbf{X})$.

4 Succinctness

It is clear from Definition 5 that the presented translation uses at most $|\mathsf{sf}(\varphi)|$ acceptance sets for Boolean combinations of $LTL_{\mathbf{FG},\mathbf{GF}}(\mathbf{X})$. We show succinctness compared to deterministic generalized Rabin automata or deterministic Streett automata, which might need an exponential sized acceptance condition for the same language, while the acceptance size only grows linearly for TELAs.

For this, we define two mutually recursive formula patterns modelling Rabin and Streett conditions:

$$\varphi_{\mathrm{R},0} = \mathbf{FG}a_0 \wedge \mathbf{GF}b_0 \qquad \varphi_{\mathrm{R},n+1} = (\mathbf{FG}a_{n+1} \wedge \mathbf{GF}b_{n+1}) \vee \varphi_{\mathrm{S},n}$$
$$\varphi_{\mathrm{S},0} = \mathbf{FG}a_0 \vee \mathbf{GF}b_0 \qquad \varphi_{\mathrm{S},n+1} = (\mathbf{FG}a_{n+1} \vee \mathbf{GF}b_{n+1}) \wedge \varphi_{\mathrm{R},n}$$

We call the subformulas $\mathbf{FG}a_j$, $\mathbf{GF}b_j$ leafs, and a set $\mathcal{L} \subseteq \mathsf{sf}(\varphi)$ of leafs a good leaf set — denoted by $\mathsf{gls}(\varphi)$ —, if it is a minimal set satisfying $\varphi_{\mathrm{R},n}$, respectively $\varphi_{\mathrm{S},n}$.

Lemma 2. *For $\varphi_{\mathrm{R},n}$ and $\varphi_{\mathrm{S},n}$ there are $\Omega(2^{\frac{n}{2}})$ good leaf sets.*

Proof. First note, that for each subformula of the form $\varphi_{\mathrm{S},k+1}$ there is a doubling of good leaf sets in $\varphi_{\mathrm{R},k}$. This comes from the conjunction of the Streett pair $\mathbf{FG}a_{k+1} \vee \mathbf{GF}b_k$ and $\varphi_{\mathrm{R},k}$: To every good leaf set of $\varphi_{\mathrm{R},k}$ one has to add either $\mathbf{FG}a_{k+1}$ or $\mathbf{GF}b_{k+1}$ to obtain a good leaf set for $\varphi_{\mathrm{S},k+1}$. On the other side, $\mathsf{gls}(\varphi_{\mathrm{R},k+1}) = \{\mathbf{FG}a_{k+1}, \mathbf{GF}b_{k+1}\} \cup \mathsf{gls}(\varphi_{\mathrm{S},n})$

Since we alternate between $\varphi_{\mathrm{R},k}$ and $\varphi_{\mathrm{S},k}$, we have $\Theta(2^{\frac{n}{2}})$ good leaf sets for $\varphi_{\mathrm{R},n}$ (resp. $\varphi_{\mathrm{S},n}$). □

W.l.o.g. we assume, that every good leaf set contains at most one subformula of the form $\mathbf{FG}\psi$. If there are more than one subformulas of this pattern, e.g. $\mathbf{FG}\psi_1$ and $\mathbf{FG}\psi_2$, one can remove both and add $\mathbf{FG}(\psi_1 \wedge \psi_2)$. Note that this transformation does not reduce the number of good leaf sets, since no good leaf set is removed, and two good leaf sets cannot be reduced to the same good leaf set.

One can easily give a bijection from a Rabin pair to a good leaf set, negate $\varphi_{\mathrm{S},n}$ to $\varphi_{\mathrm{R},n}$, and use the duality between Rabin and Streett automata. Overall, we get the following lemma:

Lemma 3. *For every $n \in \mathbb{N}$, every generalized Rabin automaton equivalent to $\varphi_{S,n}$ has at least $|\mathsf{gls}(\varphi_{S,n})|$ acceptance pairs. For every $n \in \mathbb{N}$, every Streett automaton equivalent to $\varphi_{R,n}$ has at least $|\mathsf{gls}(\varphi_{S,n})|$ acceptance pairs.*

Note that $\varphi_{S,n}$ and $\varphi_{R,n}$ are Boolean combinations of formulas from $LTL_{\mathbf{FG},\mathbf{GF}}(\mathbf{X})$. Since $\varphi_{S,n}$ and $\varphi_{R,n}$ do not contain a $\mathbf{X}$ operator, the produced automaton of our construction has exactly one state. According to Section 3.1 one can see, that the structure of the formula is directly translated into the acceptance condition. Therefore, the length of the acceptance condition is equal to $|\varphi_{S,n}|$ (resp. $|\varphi_{R,n}|$).

5 Experimental Evaluation

Our experimental evaluation is two-part: At first, we evaluate our translation by comparing the automata sizes and acceptance sizes. The second contribution in our evaluation considers probabilistic model checking with the help of automata. For every experiment, we set a time limit of 30 minutes and a memory limit of 10 GB for every process.[1]

5.1 Automata Sizes

For the comparison of the acceptance conditions, we rely on counting the number of $Fin(\cdot)$ and $Inf(\cdot)$ occurring in the acceptance condition. We compare our tool `Delag` with `Rabinizer` [9] and `ltl2tgba` of `SPOT`. Our benchmark consists of 94 LTL formulas from [30, 7, 10] where for 34 formulas `Delag` was able to translate a formula completely without using an external tool. For these formulas we do not need to rely on an external tool translating LTL to deterministic automata. Should we require external tools to translate parts of the formula, as described in Section 3.3, we use `ltl2tgba` of `SPOT` as fallback solution.

Overall, `Delag` produced automata with a minimal state space in 77 cases, followed by `ltl2tgba` with 71 formulas. For the comparison of the acceptance, `Delag` has delivered the smallest acceptance for 59 formulas, whereas `ltl2tgba` could produce an automaton with a minimal acceptance condition for 56 formulas. As it can be seen in Table 1 `Delag`, `ltl2tgba` and `Rabinizer` show roughly the same behavior, generating for 36 vs. 37 vs. 35 formulas automata with size less or equal than 3, with a slight advantage for `Delag` producing more automata of size one.

Table 1: Overview of the number of automata generated by the tools `Delag`, `ltl2tgba`, `Rabinizer` with an upper bound of states (on the left side) and with an upper bound of the number of leafs in the acceptance condition.

#States $\leqslant x$	1	2	3	4	6	10	> 10
`Delag`	9	17	36	59	75	87	7
`ltl2tgba`	6	17	37	60	78	89	5
`Rabinizer`	6	15	35	53	75	84	10

Acc. size $\leqslant x$	1	2	3	4	6	> 6
`Delag`	50	79	83	83	90	4
`ltl2tgba`	72	84	84	86	93	1
`Rabinizer`	20	34	54	67	81	13

The situation differs for the sizes of the acceptance condition: `ltl2tgba` generates 72 automata with acceptance size 1 whereas `Delag` generates 50 automata with acceptance size 1. For bigger acceptance sizes the number of generated automata are similar for `ltl2tgba` and `Delag`. In comparison, `Rabinizer` tends to produce automata with bigger acceptance sizes.

[1] All experiments were carried out on a computer with two Intel E5-2680 8-core CPUs at 2.70 GHz with 384GB of RAM running Linux.

For the formulas $\varphi_{\mathrm{R},n}$ of Section 4 the results are as expected (see Table 2). `Delag` always produces the smallest acceptance with a one state automaton, whereas the acceptance sizes of the automata produced by `Rabinizer` grow faster, e.g., for $n = 5$ and $n = 7$ `Rabinizer` produces an automaton with acceptance size 45 and 109, respectively. Both `Delag` and `Rabinizer` produce one state automata. `ltl2tgba` behaves differently: The state space size of the automata grows with n: for $n = 1$ `ltl2tgba` produces an automaton with 7 states and an acceptance size of 4, whereas for $n = 3$ the state space increased to 21889 states and an acceptance size of 20. For $n > 3$ we were not able to produce automata with `ltl2tgba`.

Table 2: Acceptance sizes for the alternating formula $\varphi_{\mathrm{R},n}$; − means time-out or mem-out.

$n =$	0	1	2	3	4	5	6	7
`Delag`	2	4	6	8	10	12	14	16
`ltl2tgba`	2	4	8	20	−	−	−	−
`Rabinizer`	2	5	7	17	19	45	47	109

For the evaluation of the history, we took the formula pattern $\varphi_{\mathcal{H},n}$:

$$\varphi_{\mathcal{H},n} = \begin{cases} \big(\mathbf{FG}(a \vee \mathbf{X}^n b)\big) \vee \varphi_{\mathcal{H},n-1} & \text{if } n \text{ is even} \\ \big(\mathbf{FG}(\neg a \vee \mathbf{X}^n b)\big) \vee \varphi_{\mathcal{H},n-1} & \text{otherwise} \end{cases}$$

Every subformula $a \vee \mathbf{X}^n b$ (or $\neg a \vee \mathbf{X}^n b$) commits the first position or the n-th position. So only two out of n positions may be fixed, and hence we can share a lot of the state space between the **FG** formulas.

The results can be found in Table 3. The state space of `ltl2tgba` grows faster than `Delag`, the former being only capable to produce automata up to $n = 5$ before hitting the memory limit. For `Rabinizer`, we were not able to produce automata for $n \geqslant 4$, since `Rabinizer` supports only a limited number of acceptance set. This shows, that the acceptance condition grows immensely.

Table 3: Automata sizes and number of acceptance sets for $\varphi_{\mathcal{H},n}$; − means time-out or mem-out.

	$n =$	0	1	2	3	4	5	6	7
`Delag`	#States	1	2	4	8	16	32	64	128
	Acc. size	1	2	3	4	5	6	7	8
`ltl2tgba`	#States	2	4	21	170	1816	22196	−	−
	Acc. size	2	2	2	2	2	2	−	−
`Rabinizer`	#States	1	2	5	11	−	−	−	−
	Acc. size	1	3	7	19	−	−	−	−

5.2 Prism Runtimes

We have implemented a routine for the analysis of MDPs in `PRISM`. Here we compare the behaviour of `PRISM` if the three tools `Delag`, `ltl2tgba` from `SPOT`, and `Rabinizer` are employed as automata generation tools. As case study we consider the IEEE 802.11 Wireless LAN Handshaking protocol. It

Table 4: PRISM runtimes (t_{MC}) for the IEEE 802.11 case study enhanced with automata sizes ($|A|$) and the number of BDD nodes in the product (BDD size $\mathcal{M} \otimes \mathcal{A}$)

Property	Delag			ltl2tgba			Rabinizer		
	$\lvert\mathcal{A}\rvert$	BDD size $\mathcal{M} \otimes \mathcal{A}$	t_{MC}	$\lvert\mathcal{A}\rvert$	BDD size $\mathcal{M} \otimes \mathcal{A}$	t_{MC}	$\lvert\mathcal{A}\rvert$	BDD size $\mathcal{M} \otimes \mathcal{A}$	t_{MC}
$\mathbf{P}_{\min}(\varphi_1)$	4	31,861	6.6 s	5	44,181	9.5 s	4	31,861	32.2 s
$\mathbf{P}_{\min}(\varphi_2)$	4	61,711	165.4 s	4	61,719	160.6 s	4	61,719	159.0 s
$\mathbf{P}_{\min}(\varphi_3)$	20	46,013	27.5 s	20	46,106	26.2 s	72	47,114	28.0 s
$\mathbf{P}_{\min}(\varphi_4)$	1	30,091	42.6 s	5	30,473	6.8 s	1	30,091	47.0 s
$\mathbf{P}_{\max}(\varphi_4)$	1	30,091	5.7 s	32	129,905	273.8 s	1	30,091	6.0 s
$\mathbf{P}_{\min}(\varphi_5)$	4	61,711	120.9 s	21	65,504	91.6 s	4	61,719	125.5 s
$\mathbf{P}_{\max}(\varphi_5)$	4	61,711	152.7 s	40	182,133	861.6 s	4	61,719	165.1 s

describes a resolving mechanism to stop interference if two stations want to send a message at the same time. The key trick is, that all participating stations listen to interference, and if a message has become garbled, the stations waits a random amount of time (limited by an upper bound called Backoff) and then tries to resend the message. We used the following properties:

- "If a message from sender i has been garbled, it will be sent correctly in the future" $\varphi_1 = \bigwedge_{1 \le i \le n} \mathbf{G}(\texttt{garbled}_i \rightarrow \mathbf{F}\,\texttt{correct}_i)$
- "Every sender sends at least one message correctly." : $\varphi_2 = \bigwedge_{1 \le i \le n} \mathbf{F}\,\texttt{correct}_i$
- "The first time every station wants to send, the channel remains free for k steps" $\varphi_3 = \bigwedge_{1 \le i \le n} \texttt{wait}_i\,\mathbf{U}(\texttt{wait}_i \wedge \mathbf{G}^{\le k}\,\texttt{free})$ where $\mathbf{G}^{\le k}\texttt{free} = \texttt{free} \wedge \mathbf{X}\texttt{free} \wedge \ldots \wedge \mathbf{X}^n\,\texttt{free}$
- "Every station, that wants to send a message infinitely often, is able to send a message correctly infinitely often " : $\varphi_4 = \bigwedge_{1 \le i \le n}(\mathbf{G}\,\mathbf{F}\,\texttt{wait}_i) \rightarrow (\mathbf{G}\,\mathbf{F}\,\texttt{correct}_i)$
- "Every station satisfies both the reachability formula φ_2 and the fairness formula φ_4" $\varphi_5 = \left(\bigwedge_{1 \le i \le n} \mathbf{F}\,\texttt{correct}_i\right) \wedge \left(\bigwedge_{1 \le i \le n}(\mathbf{G}\,\mathbf{F}\,\texttt{wait}_i) \rightarrow (\mathbf{G}\,\mathbf{F}\,\texttt{correct}_i)\right)$

Every property can be translated directly by Delag without external tools, except φ_1, for which we translate the subformulas $\mathbf{G}(\texttt{garbled}_i \rightarrow \mathbf{F}\,\texttt{correct}_i)$ with ltl2tgba and then build the product. So φ_1 should be seen as a benchmark for the product construction.

For all properties we asked for the minimal ($\mathbf{P}_{\min}(\cdot)$) or maximal ($\mathbf{P}_{\max}(\cdot)$) probability of the IEEE 802.11 handshaking model with two stations and a Backoff of at most 3 to satisfy the property. If a formula has a window length (e.g. $\mathbf{G}^{\le k}$) we uniformly choose $k = 6$. Table 4 lists some measured time values and automata/product sizes. All PRISM experiments were carried out with the hybrid engine, an engine that combines symbolic and explicit data structures offering a good compromise.

First, the generation time for every automaton was below 1.0 s, except for Rabinizer at $\mathbf{P}_{\min}(\varphi_3)$ where it was 1.8 s. In 3 cases PRISM in combination with Delag was the fastest. For $\mathbf{P}_{\min}(\varphi_4)$ ltl2tgba took only 6.8 s in comparison to 42.6 s for Delag despite the smaller automaton, since one heuristic applied for ltl2tgba that did not apply for Delag: For the analysis of maximal end-components (MEC) we checked always at first, if the whole MEC satisfies the acceptance condition, and only if not, we look

for accepting sub-end-components within the MEC. For `ltl2tgba` the whole MEC was accepting, but for `Delag` one had to search for an accepting sub-end-component. Since in a symbolic representation SCC enumeration is costly, `ltl2tgba` was much faster.

In general, one can see, that `Delag` produced every time the smallest automaton, that also results in the smallest number of BDD nodes in the product and comparatively small model checking times.

We have checked two more properties in the full version [1] as well as included a comparison with the standard approach of `PRISM` that uses an own implementation of `ltl2ba` [12] and Safra's determinisation [26] and delivers automata with state-based acceptance.

6 Conclusion

We presented a general framework based on the product construction and specialised translations for fragments of LTL to build deterministic Emerson-Lei automata. In particular, for the important fairness fragment we established an efficient construction, where the state space only depends on the nesting depth of **X**, and all of the complexity is shifted to the acceptance condition. The general construction applies a range of additional optimisations: such as pushing temporal operators down the syntax tree, piggybacking to reduce the number of acceptance sets and sharing of equal automata parts. In particular our history buffer approach reduces the state space, since the buffer can be shared between automata for different subformulas. If a formula does not belong to one of our explicitly supported fragments, we can run an external LTL to deterministic automaton translator and incorporate the resulting automaton via product construction and lifting.

Benchmarking this approach has shown the potential of our method. Standard benchmarks highlight the potential of allowing more complex acceptance conditions, our tool had a slight advantage in the state space over `SPOT`. Those results also reflect in the area of probabilistic model checking, where we analysed the IEEE 802.11 Handshaking protocol.

However, the heuristics presented here are not complete, and this approach should be understood as a framework. So, one direction for future work is to add more explicitly supported LTL fragments. Another point would be to analyse the subformulas, which cannot be translated directly and choose an external tool, that behaves well for these specific subformulas. For example, it is well-known, that obligation LTL formulas can be translated to weak DBA, and then efficiently minimised. This is implemented in `SPOT`. Another direction one could take a deeper look into, is to start with a non-deterministic Büchi automaton, and try to find small deterministic automaton with a complex acceptance condition. Of course, general methods to shrink the state space like bisimulation could be also applied. Also, the particular ingredients of our transformation could optimised further, e.g. the history could be allocated dynamically, and therefore reduce the state space even further without increasing the acceptance condition complexity.

Acknowledgments. The authors want to thank the anonymous reviewers for the constructive feedback.

References

[1] `http://wwwtcs.inf.tu-dresden.de/ALGI/PUB/GandALF17-EL`. Website with additional material.

[2] Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker & Jan Strejček (2015): *The Hanoi Omega-Automata Format*. In: *27th International Conference on Computer Aided Verification (CAV)*, *Lecture Notes in Computer Science* 9206, Springer, pp. 479–486, doi:10.1007/978-3-319-21690-4_31.

[3] Tomás Babiak, Frantisek Blahoudek, Mojmír Kretínský & Jan Strejcek (2013): *Effective Translation of LTL to Deterministic Rabin Automata: Beyond the (F, G)-Fragment*. In: *11th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, *Lecture Notes in Computer Science* 8172, Springer, pp. 24–39, doi:10.1007/978-3-319-02444-8_4.

[4] Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller & James Worrell (2016): *Markov Chains and Unambiguous Büchi Automata*. In: *Proc. of the 28th International Conference on Computer Aided Verification (CAV) - Part I*, *Lecture Notes in Computer Science* 9779, Springer, pp. 23–42, doi:10.1007/978-3-319-41528-4_2.

[5] Costas Courcoubetis & Mihalis Yannakakis (1995): *The Complexity of Probabilistic Verification*. *J. ACM* 42(4), pp. 857–907, doi:10.1145/210332.210339.

[6] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault & Laurent Xu (2016): *Spot 2.0 - A Framework for LTL and ω-Automata Manipulation*. In: *14th Internation Symposium on Automated Technology for Verification and Analysis (ATVA)*, *Lecture Notes in Computer Science* 9938, pp. 122–129, doi:10.1007/978-3-319-46520-3_8.

[7] Matthew B. Dwyer, George S. Avrunin & James C. Corbett (1999): *Patterns in Property Specifications for Finite-State Verification*. In: *21th International Conference on Software Engineering (ICSE)*, ACM, pp. 411–420, doi:10.1145/302405.302672.

[8] Javier Esparza, Jan Kretínský, Jean-François Raskin & Salomon Sickert (2017): *From LTL and Limit-Deterministic Büchi Automata to Deterministic Parity Automata*. In: *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, *Lecture Notes in Computer Science* 10205, pp. 426–442, doi:10.1007/978-3-662-54577-5_25.

[9] Javier Esparza, Jan Kretínský & Salomon Sickert (2016): *From LTL to deterministic automata - A safraless compositional approach*. *Formal Methods in System Design* 49(3), pp. 219–271, doi:10.1007/s10703-016-0259-2.

[10] Kousha Etessami & Gerard Holzmann (2000): *Optimizing Büchi Automata*. In: *11th International Conference on Concurrency Theory (CONCUR)*, *Lecture Notes in Computer Science* 1877, Springer, pp. 153–167, doi:10.1007/3-540-44618-4_13.

[11] Dana Fisman & Yoad Lustig (2015): *A Modular Approach for Büchi Determinization*. In: *26th International Conference on Concurrency Theory (CONCUR)*, *LIPIcs* 42, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 368–382, doi:10.4230/LIPIcs.CONCUR.2015.368.

[12] Paul Gastin & Denis Oddoux (2001): *Fast LTL to Büchi Automata Translation*. In: *13th International Conference on Computer Aided Verification (CAV)*, *Lecture Notes in Computer Science* 2102, Springer, pp. 53–65, doi:10.1007/3-540-44585-4_6.

[13] Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini & Lijun Zhang (2015): *Lazy Probabilistic Model Checking without Determinisation*. In: *26th International Conference on Concurrency Theory (CONCUR)*, *LIPIcs* 42, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 354–367, doi:10.4230/LIPIcs.CONCUR.2015.354.

[14] Thomas A. Henzinger & Nir Piterman (2006): *Solving Games Without Determinization*. In: *20th Annual Conference on Computer Science Logic (CSL)*, *Lecture Notes in Computer Science* 4207, Springer, pp. 395–410, doi:10.1007/11874683.

[15] Detlef Kähler & Thomas Wilke (2008): *Complementation, Disambiguation, and Determinization of Büchi Automata Unified*. In: *35th International Colloquium on Automata, Languages and Programming (ICALP)*, *Lecture Notes in Computer Science* 5125, Springer, pp. 724–735, doi:10.1007/978-3-540-70575-8_59.

[16] Joachim Klein & Christel Baier (2006): *Experiments with deterministic ω-automata for formulas of linear temporal logic*. *Theoretical Computer Science* 363(2), pp. 182–195, doi:10.1016/j.tcs.2006.07.022.

[17] Joachim Klein & Christel Baier (2007): *On-the-Fly Stuttering in the Construction of Deterministic ω-Automata*. In: *12th International Conference on Implementation and Application of Automata (CIAA)*, *Lecture Notes in Computer Science* 4783, Springer, pp. 51–61, doi:10.1007/978-3-540-76336-9_7.

[18] Joachim Klein, David Müller, Christel Baier & Sascha Klüppelholz (2014): *Are Good-for-Games Automata Good for Probabilistic Model Checking?* In: *8th International Conference on Language and Automata Theory and Applications (LATA)*, *Lecture Notes on Computer Science* 8370, Springer, pp. 453–465, doi:10.1007/978-3-319-04921-2_37.

[19] Orna Kupferman, Nir Piterman & Moshe Y. Vardi (2006): *Safraless Compositional Synthesis*. In: *18th International Conference on Computer Aided Verification (CAV)*, *Lecture Notes in Computer Science* 4144, Springer, pp. 31–44, doi:10.1007/11817963_6.

[20] Yong Li, Lei Song, Yuan Feng & Lijun Zhang (2016): *Verify LTL with Fairness Assumptions Efficiently*. In: *23rd International Symposium on Temporal Representation and Reasoning, (TIME)*, IEEE Computer Society, pp. 41–50, doi:10.1109/TIME.2016.12.

[21] Andreas Morgenstern & Klaus Schneider (2008): *From LTL to Symbolically Represented Deterministic Automata*. In: *9th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, *Lecture Notes in Computer Science* 4905, Springer, pp. 279–293, doi:10.1007/978-3-540-78163-9_24.

[22] Andreas Morgenstern & Klaus Schneider (2010): *Exploiting the Temporal Logic Hierarchy and the Non-Confluence Property for Efficient LTL Synthesis*. In: *First Symposium on Games, Automata, Logics, and Formal Verification (GandALF)*, *Electronic Proceedings in Theoretical Computer Science* 25, pp. 89–102, doi:10.4204/EPTCS.25.11.

[23] Nir Piterman (2007): *From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata*. *Logical Methods in Computer Science* 3(3), doi:10.2168/LMCS-3(3:5)2007.

[24] Roman R. Redziejowski (2012): *An Improved Construction of Deterministic Omega-automaton Using Derivatives*. *Fundam. Inform.* 119(3-4), pp. 393–406, doi:10.3233/FI-2012-744.

[25] S. Safra & M. Y. Vardi (1989): *On Ω-automata and Temporal Logic*. In: *21th Annual Symposium on Theory of Computing*, STOC'89, ACM, doi:10.1145/73007.73019.

[26] Shmuel Safra (1988): *On the complexity of ω-automata*. In: *29th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, pp. 319–327, doi:10.1109/SFCS.1988.21948.

[27] Sven Schewe (2009): *Tighter Bounds for the Determinisation of Büchi Automata*. In: *12th International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*, Lecture Notes in Computer Science, Springer, pp. 167–181, doi:10.1007/978-3-642-00596-1_13.

[28] Salomon Sickert, Javier Esparza, Stefan Jaax & Jan Kretínský (2016): *Limit-Deterministic Büchi Automata for Linear Temporal Logic*. In: *28th International Conference on Computer Aided Verification (CAV)*, *Lecture Notes in Computer Science* 9780, Springer, pp. 312–332, doi:10.1007/978-3-319-41540-6_17.

[29] Salomon Sickert & Jan Kretínský (2016): *MoChiBA: Probabilistic LTL Model Checking Using Limit-Deterministic Büchi Automata*. In: *14th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, *Lecture Notes in Computer Science* 9938, Springer, pp. 130–137, doi:10.1007/978-3-319-46520-3_9.

[30] Fabio Somenzi & Roderick Bloem (2000): *Efficient Büchi Automata from LTL Formulae*. In: *12th International Conference on Computer Aided Verification (CAV)*, *Lecture Notes in Computer Science* 1855, Springer, pp. 248–263, doi:10.1007/10722167_21.

[31] Moshe Y. Vardi (1985): *Automatic verification of probabilistic concurrent finite-state programs*. In: *26th IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, pp. 327–338, doi:10.1109/SFCS.1985.12.