Limit-Deterministic Büchi Automata for Linear Temporal Logic[⋆]

Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský

Technische Universität München

Abstract. Limit-deterministic Büchi automata can replace deterministic Rabin automata in probabilistic model checking algorithms, and can be significantly smaller. We present a direct construction from an LTL formula φ to a limit-deterministic Büchi automaton. The automaton is the combination of a non-deterministic component, guessing the set of eventually true $\mathbf{G}$-subformulas of φ, and a deterministic component verifying this guess and using this information to decide on acceptance. Contrary to the indirect approach of constructing a non-deterministic automaton for φ and then applying a semi-determinisation algorithm, our translation is compositional and has a clear logical structure. Moreover, due to its special structure, the resulting automaton can be used not only for qualitative, but also for quantitative verification of MDPs, using *the same* model checking algorithm as for deterministic automata. This allows one to reuse existing efficient implementations of this algorithm without any modification. Our construction yields much smaller automata for formulas with deep nesting of modal operators and performs at least as well as the existing approaches on general formulas.

1 Introduction

Translating Linear Temporal Logic (LTL) formulas into ω-automata is a fundamental problem of formal verification, which has been studied in depth. In the automata-theoretic approach to model checking, after computing the automaton for a formula one constructs its product with the state space of the system under consideration and analyses it. Since the product has up to $N \cdot m$ states, where N and m are the number of states of the system and the automaton, respectively, and typically $N \gg m$, it is important to construct automata as small as possible: even a small reduction of m can lead to a much larger reduction of $N \cdot m$.

Since non-deterministic ω-automata are typically much smaller than deterministic ones, for standard LTL model checking one translates formulas into non-deterministic Büchi automata. However, this is no longer possible for probabilistic model checking, and the standard approach is to use deterministic Rabin automata (DRA) instead—this is for instance the approach of the PRISM tool [3, 22]. Translations of LTL into DRA have been thoroughly studied. Classical translations take a detour through Büchi automata as intermediate step [25, 24, 26], while more recent ones are direct translations [11, 19].

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

It has been known for a long time that automata for probabilistic verification do not need to be fully deterministic: the automata-theoretic approach still works if *restricted* forms of non-determinism are allowed. For probabilistic verification of Markov chains one can use unambiguous Büchi automata (separated UBA [7], or even non-separated UBA [2]). The translation from LTL to separated UBA involves a single exponential blowup, while the translation to DRA is known to be double exponential. However, UBA cannot be used for the verification of Markov Decision Processes (MDPs). For qualitative verification of MDPs[1] one can use *limit-deterministic* Büchi automata (LDBA) [27, 6] (also known as *semi-deterministic* or *deterministic-in-the-limit*). For quantitative verification of MDPs, limit-deterministic automata are not sufficient in general. However, recently [14], a more complex algorithm for probabilistic model checking was presented, considering products of the system and its parts with several different automata, including LDBA.

The translation LTL→LDBA is double exponential, and so in principle as expensive as the translation to DRA in the worst case. However, it is easy to find examples where the LDBA is much smaller than the DRA; in particular, in [16] it is shown that the $\mathrm{LTL}_{\setminus \mathbf{GU}}$ fragment of LTL can be translated to LDBAs with a single-exponential blowup, while the translation to DRA is still double exponential. Further, efficient procedures for LDBA complementation exist [4].

In this paper, we give a compositional translation from *full* LTL to LDBAs, based on the one from LTL to DRAs recently presented in [11]. We then show that, due to the special form of the resulting LDBAs, the translation can also be used for *quantitative* model checking of MDPs, and in fact by means of *the same algorithm* as for DRAs. That is, in order to compute the maximal probability that an MDP M satisfies a formula φ we can just construct the product of M and the LDBA for φ obtained by our translation, and compute the maximal probability of reaching an accepting end component [3].

The translation of [11] becomes much simpler with LDBAs as target, instead of DRAs. In order to check if a word w satisfies a formula φ, our LDBAs use their restricted non-determinism to guess the set of $\mathbf{G}$-subformulas of φ that are eventually satisfied by w (i.e., the subformulas $\mathbf{G}\varphi$ such that $w \models \mathbf{FG}\psi$), and the point at which all these subformulas have already become true. At this point the LDBA enters its deterministic component to check that the guess is correct, and that $w \models \varphi$ holds under the assumption that the guess is correct. We show that our translation produces LDBA of at most double exponential size, and exhibit a family of LTL formulas for which the smallest LDBA reaches this double exponential bound.

We conclude the paper with an experimental evaluation of an implementation of our construction on a large set of benchmarks. We compare the size of the generated LDBA with the size of the DRA provided by the Rabinizer tool, the DRA constructed by LTL2DSTAR, and the LDBA produced by the procedure of [6]: translate the formula into a Büchi automaton, apply the translation of Büchi automata to LDBA described in [6], and simplify the result. For the comparison we use the LTL3BA tool [1] and SPOT [8].

[1] Recall that qualitative verification checks if the property holds with probability 1.

Outline. Sections 2 and 3 contain preliminaries. Section 4 presents an intuitive overview of the translation by means of an example. Section 5 formally defines the translation, Section 6 describes several optimisations, and Section 7 gives the complexity bounds. Quantitative verification is discussed in Section 8. Experimental results are presented in Section 9. Section 10 concludes and discusses future work.

2 Preliminaries

2.1 Linear Temporal Logic

We use a slightly unusual syntax for LTL. We consider formulas without negations and without the *Release* operator $\mathbf{R}$— the dual of the *Until* operator $\mathbf{U}$— but with both $\mathbf{F}$ and $\mathbf{G}$. Formulas in the usual syntax are transformed into equivalent formulas in our syntax by pushing negations inside, and—in the absence of $\mathbf{R}$— using the equivalence $\neg(\varphi\mathbf{U}\psi) = (\neg\psi\mathbf{U}(\neg\psi \wedge \neg\varphi)) \vee \mathbf{G}\neg\psi$. This may cause the formula to grow exponentially, when formulas are represented by their syntax trees. However, if they are represented by their syntax DAGs, then the transformation only causes a linear blowup.

Definition 1 (LTL). *A formula of LTL in* negation normal form *is given by the syntax:*

$$\varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi\mathbf{U}\varphi$$

where $a \in Ap$. An ω-word w is an infinite sequence of letters $w[0]w[1]w[2]\dots$. We denote the infinite suffix $w[i]w[i+1]\dots$ by w_i. The satisfaction relation $\models$ between ω-words and formulas is inductively defined as follows:

$$\begin{array}{lll lll}
w \models \mathbf{tt} & & w \not\models \mathbf{ff} & w \models \mathbf{X}\varphi & \mathit{iff} & w_1 \models \varphi \\
w \models a & \mathit{iff} & a \in w[0] & w \models \mathbf{F}\varphi & \mathit{iff} & \exists k.\, w_k \models \varphi \\
w \models \neg a & \mathit{iff} & a \notin w[0] & w \models \mathbf{G}\varphi & \mathit{iff} & \forall k.\, w_k \models \varphi \\
w \models \varphi \wedge \psi & \mathit{iff} & w \models \varphi \textit{ and } w \models \psi & w \models \varphi\mathbf{U}\psi & \mathit{iff} & \exists k.\, w_k \models \psi \textit{ and} \\
w \models \varphi \vee \psi & \mathit{iff} & w \models \varphi \textit{ or } w \models \psi & & & \forall 0 \leq j < k.\, w_j \models \varphi
\end{array}$$

Given two formulas φ and ψ, we denote by $\varphi[\Phi/\psi]$ the result of substituting ψ for each maximal occurrence of a formula of Φ in φ (an occurrence is maximal if it is not a subformula of another occurrence). For example, $\mathbf{G}(a \vee \mathbf{G}b)[\{\mathbf{G}(a \vee \mathbf{G}b), \mathbf{G}b\}/\mathbf{tt}] = \mathbf{tt}$. Two formulas are *equivalent* if they are satisfied by the same words. We under-approximate this using propositional equivalence.

Definition 2 (Propositional Equivalence). *A subformula ψ of φ is called* proper*, if the root of its syntax tree is labelled by either a, $\neg a$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{U}$ or $\mathbf{X}$. Given a formula φ, we assign to it a propositional formula φ_P as follows: replace every maximal proper subformula ψ by a propositional variable x_ψ. Two formulas φ, ψ are* propositionally equivalent*, denoted $\varphi \equiv_P \psi$, iff φ_P and ψ_P are equivalent formulas of propositional logic. We denote by $[\varphi]_P$ the set of all formulas propositional equivalent to φ.*

For example, if $\varphi = \mathbf{X}b \vee (\mathbf{G}(a \vee \mathbf{X}b) \wedge \mathbf{X}b)$ with $\psi_1 = \mathbf{X}b$ and $\psi_2 = \mathbf{G}(a \vee \mathbf{X}b)$, then $\varphi_P = x_{\psi_1} \vee (x_{\psi_2} \wedge x_{\psi_1}) \equiv x_{\psi_1}$. Thus $\mathbf{X}b$ is propositionally equivalent to φ and $\mathbf{X}b \in [\varphi]_P$.

2.2 Formula Expansion

Our translation relies on the "After Function" $af(\varphi, w)$, read "φ after w" [11]. Intuitively, φ holds for ww' iff $af(\varphi, w)$ holds "after reading w", that is, if $w' \models af(\varphi, w)$.

Definition 3. *Let φ be a formula and $\nu \in 2^{Ap}$ a single letter. $af(\varphi, \nu)$ is then defined as follows:*

$$\begin{aligned}
&af(\mathbf{tt}, \nu) = \mathbf{tt} && af(\varphi \wedge \psi, \nu) = af(\varphi, \nu) \wedge af(\psi, \nu) \\
&af(\mathbf{ff}, \nu) = \mathbf{ff} && af(\varphi \vee \psi, \nu) = af(\varphi, \nu) \vee af(\psi, \nu) \\
&af(a, \nu) = \begin{cases} \mathbf{tt} \text{ if } a \in \nu \\ \mathbf{ff} \text{ if } a \notin \nu \end{cases} && \begin{aligned} af(\mathbf{X}\varphi, \nu) &= \varphi \\ af(\mathbf{G}\varphi, \nu) &= af(\varphi, \nu) \wedge \mathbf{G}\varphi \end{aligned} \\
&af(\neg a, \nu) = \begin{cases} \mathbf{ff} \text{ if } a \in \nu \\ \mathbf{tt} \text{ if } a \notin \nu \end{cases} && \begin{aligned} af(\mathbf{F}\varphi, \nu) &= af(\varphi, \nu) \vee \mathbf{F}\varphi \\ af(\varphi\mathbf{U}\psi, \nu) &= af(\psi, \nu) \vee (af(\varphi, \nu) \wedge \varphi\mathbf{U}\psi) \end{aligned}
\end{aligned}$$

Furthermore, we generalize the definition to finite words: $af(\varphi, \epsilon) = \varphi$; and $af(\varphi, \nu w) = af(af(\varphi, \nu), w)$ for every $\nu \in 2^{Ap}$ and every finite word w. Finally, we define the set of from φ reachable formulas as $Reach(\varphi) = \{[\psi]_P \mid \exists w.\ \psi = af(\varphi, w)\}$.

Example 1. Let $Ap = \{a, b, c\}$ and $\varphi = a \vee (b\ \mathbf{U}\ c)$. We have $af(\varphi, \{a\}) = \mathbf{tt}$ $af(\varphi, \{b\}) = (b\ \mathbf{U}\ c)$, $af(\varphi, \{c\}) = \mathbf{tt}$, and $af(\varphi, \emptyset) = \mathbf{ff}$.

The following lemmas show that af has indeed the claimed property, and others.

Lemma 1 ([11], Lem. 7). *Let φ be a formula, and let $ww' \in (2^{Ap})^\omega$ be an arbitrary word. Then $ww' \models \varphi$ iff $w' \models af(\varphi, w)$.*

Lemma 2 ([11], Lem. 11). *Let φ be a $\mathbf{G}$-free formula and let w be a word. Then $w \models \varphi$ iff there exists $i > 0$ such that $af(\varphi, w_{0j}) \equiv_P \mathbf{tt}$ for every $j \geq i$.*

We now show that $Reach(\varphi)$ — a building block for the construction — is finite, and contains at most a double exponential number of elements.

Lemma 3. *For every formula φ and every finite word $w \in (2^{Ap})^*$:*

(1) $af(\varphi, w)$ is a boolean combination of proper subformulas of φ.
(2) If φ has n proper subformulas, then $Reach(\varphi)$ has at most size 2^{2^n}.

Proof. (1) By definition, every formula is a boolean combination of its proper subformulas. So it suffices to prove that every proper subformula of $af(\varphi, w)$ is also a proper subformula of φ. For $w = \nu$ this follows by an easy induction on φ, and for an arbitrary w by induction on $|w|$.

(2) By (1), every equivalence class $[\psi]_P \in \mathit{Reach}(\varphi)$ can be uniquely identified with a Boolean function over n variables, one for each proper subformula of φ. Since there are 2^{2^n} Boolean functions over n variables, we have at most so many equivalence classes. $\square$

Remark 1. It is easy to show by induction that $\varphi \equiv_P \psi$ implies $\mathit{af}(\varphi, w) \equiv_P \mathit{af}(\psi, w)$ for every finite word w. We extend af to equivalence classes by defining $\mathit{af}([\varphi]_P, w) := [\mathit{af}(\varphi, w)]_P$. Sometimes we abuse language and identify a formula and its equivalence class. For example, we write "the states of the automaton are pairs of formulas" instead of "pairs of equivalence classes of formulas".

3 Limit-deterministic Büchi Automata

For convenience, we use Büchi automata with an accepting set of transitions, instead of an accepting set of places. We also consider generalized Büchi automata with several sets of accepting transitions. It is well known that all these classes accept the ω-regular languages and there are polynomial-time translations between them.

Definition 4 (Transition-based Generalized Büchi Automata). *A generalized transition-based Büchi automaton (TGBA) is a tuple $\mathcal{B} = (\Sigma, Q, \Delta, q_0, \alpha)$ where Σ is an alphabet, Q is a finite set of states, $\Delta \colon Q \times \Sigma \to 2^Q$ is a transition function, q_0 is the initial state, and $\alpha = \{F_1, F_2, \ldots F_n\}$ with $F_i \subseteq Q \times \Sigma \times Q$ is an accepting condition.*

A run r of a TGBA $\mathcal{B}$ on the ω-word w is an infinite sequence of transitions $r = (q_0, w[0], q_1)(q_1, w[1], q_2) \ldots$ respecting the transition function, i.e. $r[i] \in \Delta$ for every $i \geq 0$. We denote by $\inf(r)$ the set of transitions occurring infinitely often in the run. A run is called accepting if for each set of transitions $F \in \alpha$ there is at least one transition in the run occurring infinitely often, i.e. if $\inf(r) \cap F \neq \emptyset$. An infinite word w is accepted by $\mathcal{B}$ and is in the language $L(\mathcal{B})$ if there exists an accepting run r for w.

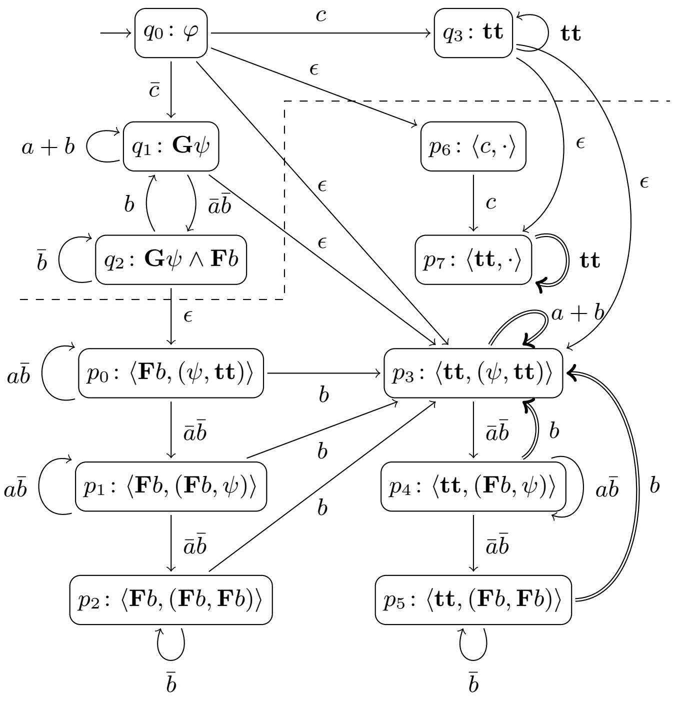
Intuitively, a TGBA is limit-deterministic if it can be split into a non-deterministic component without accepting transitions, and a deterministic component. The automaton can only accept by "jumping" from the non-deterministic to the deterministic component, but after the jump must stay in the deterministic component forever.

Definition 5 (Limit-Determinism). *A TGBA $\mathcal{B} = (\Sigma, Q, \Delta, q_0, \alpha)$ is* limit-deterministic *if Q can be partitioned into two disjoint sets $Q = Q_\mathcal{N} \uplus Q_\mathcal{D}$, s.t.*

1. *$\Delta(q, \nu) \subseteq Q_\mathcal{D}$ and $|\Delta(q, \nu)| = 1$ for every $q \in Q_\mathcal{D}$, $\nu \in \Sigma$ and*
2. *$F \subseteq Q_\mathcal{D} \times \Sigma \times Q_\mathcal{D}$ for all $F \in \alpha$*

4 Overview of the Construction

We first explain the main ideas underlying our construction on the formula $\varphi = c \vee \mathbf{XG}(a \vee \mathbf{F}b)$, and then show how to generalise them to arbitrary formulas.



$q_0 : \varphi$
$q_3 : \mathbf{tt}$
$\mathbf{tt}$
c
ϵ
$\bar{c}$
$q_1 : \mathbf{G}\psi$
$a+b$
$p_6 : \langle c, \cdot\rangle$
b
$\bar{a}\bar{b}$
$q_2 : \mathbf{G}\psi \wedge \mathbf{F}b$
$\bar{b}$
$p_7 : \langle \mathbf{tt}, \cdot\rangle$
$p_0 : \langle \mathbf{F}b, (\psi, \mathbf{tt})\rangle$
$a\bar{b}$
$p_3 : \langle \mathbf{tt}, (\psi, \mathbf{tt})\rangle$
$p_1 : \langle \mathbf{F}b, (\mathbf{F}b, \psi)\rangle$
$p_4 : \langle \mathbf{tt}, (\mathbf{F}b, \psi)\rangle$
$p_2 : \langle \mathbf{F}b, (\mathbf{F}b, \mathbf{F}b)\rangle$
$p_5 : \langle \mathbf{tt}, (\mathbf{F}b, \mathbf{F}b)\rangle$

Fig. 1. Automaton $\mathcal{A}$ for $\varphi = c \vee \mathbf{XG}(a \vee \mathbf{F}b)$. Non-accepting sinks ($\langle \mathbf{ff}, \cdot\rangle$) and transitions to them have been removed. The initial component is above the dashed line, the accepting component below. States in the lower part are tuples of an auxiliary monitor and $\mathbf{G}$-monitors.

We abbreviate $\psi := (a \vee \mathbf{F}b)$, and so we write $\varphi = c \vee \mathbf{XG}\psi$. The complete automaton for the formula is shown in Figure 1.

Each state of the automaton for φ is labelled with a formula (the state is the equivalence class of this formula w.r.t. propositional equivalence). The words accepted from the state are exactly those satisfying the formula. We describe the initial and accepting components of the automaton, separated in Figure 1 by the dashed line.

The initial component. The states of the initial component are the formulas of $\mathit{Reach}(\varphi)$, with φ as initial state. The non-ϵ transitions are given by the af-function: for every state φ' and letter ν there is a transition labelled by ν leading to $\mathit{af}(\varphi', \nu)$. With these transitions the component keeps track of the formula that must be satisfied by the rest of the word.

The only non-determinism is introduced by the "ϵ-jumps" into the accepting component. Imagine the automaton is currently at state φ'. The automaton has to check that φ' holds (more formally, that the rest of the run satisfies φ'). Intuitively, taking an ϵ-jump corresponds to picking a subset $\mathcal{G}$ of the $\mathbf{G}$-subformulas of φ', guessing that they currently hold, and guessing further that

φ' holds *even if no other* $\mathbf{G}$*-subformula becomes true in the future.* In order to see how the automaton can check this guess, we introduce some notation.

Definition 6. *Given a formula φ and a set $\mathcal{G}$ of $\mathbf{G}$-formulas, we denote $\varphi[\mathcal{G}]$ the result of substituting* $\mathbf{tt}$ *for every* $\mathbf{G}$*-subformula of $\mathcal{G}$ and* $\mathbf{ff}$ *for every other* $\mathbf{G}$*-subformula.*

We claim that after the jump the accepting component can check the guess by checking if (a) $\mathbf{G}\big(\psi[\mathcal{G}]\big)$ holds for every $\mathbf{G}\psi \in \mathcal{G}$, and (b) $\varphi'[\mathcal{G}]$ holds.

Indeed, if $\mathbf{G}\psi \in \mathcal{G}$ holds now, then it always holds in the future, and so it can be replaced by $\mathbf{tt}$. Similarly, if $\mathbf{G}\psi \notin \mathcal{G}$, then φ' should hold even if $\mathbf{G}\psi$ never holds in the future, which—since formulas are in negation normal form—is the case if φ' holds even after $\mathbf{G}\psi$ is replaced by $\mathbf{ff}$.

The crucial point now is that the formulas of (a) are of the form $\mathbf{G}\psi'$, where ψ' is $\mathbf{G}$-free, and the formula in (b) contains no occurrence of $\mathbf{G}$ at all. So for the accepting component (described below) it suffices to find deterministic automata for such formulas.

As a concrete example, consider the two ϵ-transitions of Figure 1 leaving the state q_0. Since $\mathbf{G}\psi$ is the only $\mathbf{G}$-subformula of φ, the two possible choices for $\mathcal{G}$ are $\mathcal{G} = \{\mathbf{G}\psi\}$ and $\mathcal{G} = \emptyset$. In the first case, the ϵ-transition for $\mathcal{G} = \{\mathbf{G}\psi\}$ must lead to a state in charge of checking that (a) $\mathbf{G}(\psi[\mathcal{G}]) = \mathbf{G}(a \vee \mathbf{F}b)$ holds, and (b) $\varphi[\mathbf{G}\psi/\mathbf{tt}] = c \vee \mathbf{X}\mathbf{tt} \equiv_P \mathit{true}$ holds (in this case (b) is trivial). This is the state p_3, whose successors are the part of the accepting component in charge of checking $\mathbf{G}(a \vee \mathbf{F}b)$. The ϵ-transition for $\mathcal{G} = \emptyset$ must lead to a state in charge of checking $\varphi[\mathbf{G}\psi/\mathbf{ff}] = c \vee \mathbf{X}\mathbf{ff} \equiv_P c$ (if this holds, then φ' holds, independently of whether $\mathbf{G}\psi$ holds or not). This is state p_6.

The accepting component. The accepting component consists of several subcomponents, one for each set $\mathcal{G}$ of $\mathbf{G}$-formulas. In Figure 1 there are two subcomponents, one with states $\{p_0, \ldots, p_5\}$ for $\mathcal{G} = \{\mathbf{G}\psi\}$, and the other with states $\{p_6, p_7\}$ for $\mathcal{G} = \emptyset$. A subcomponent is a product of deterministic automata: a $\mathbf{G}$-*monitor* for every formula $\mathbf{G}\psi \in \mathcal{G}$, in charge of checking $\mathbf{G}\big(\psi[\mathcal{G}]\big)$, and an *auxiliary monitor* for each state φ' of the initial component, in charge of checking $\varphi'[\mathcal{G}]$. Consider for instance the subcomponent with states $\{p_0, \ldots, p_5\}$. It is the product of a three-state $\mathbf{G}$-monitor for $\mathbf{G}\psi$, and a two-state auxiliary monitor for checking $\mathbf{F}b$.

Since $\varphi'[\mathcal{G}]$ contains no occurrence of $\mathbf{G}$, it is easy to give a deterministic auxiliary monitor, and we do so in Section 5.2. For the $\mathbf{G}$-*monitor* for $\mathbf{G}\big(\psi[\mathcal{G}]\big)$ we use a breakpoint construction. Intuitively, the monitor must check that every suffix of the monitored word satisfies ψ', and so after each step it receives ψ' as new "proof obligation". Its states are pairs of formulas (ρ_1, ρ_2). At state (ρ_1, ρ_2) the monitor is currently taking care of the proof obligation ρ_1, and has put ρ_2 on hold. Initially $\rho_1 = \psi$ and $\rho_2 = \mathbf{tt}$. Transitions of the form $\delta((\rho_1, \rho_2), a) = (\mathit{af}(\rho_1, a), \mathit{af}(\rho_2, a) \wedge \psi')$. update the proof obligations according to the af-function, adding ψ' to the proof obligation on hold. If $\rho_1 = \mathbf{tt}$ then the current proof obligation can be discarded, and the monitor can take care of the one on hold; this is done by a transition $\delta((\mathbf{tt}, \rho_2), a) = (\mathit{af}(\rho_2, a) \wedge \psi', \mathbf{tt})$.

These “discarding” transitions are accepting. If they are taken infinitely often, then all of the infinitely many proof obligations are eventually discarded.

5 Construction

For the formal presentation of the construction, let φ be a fixed formula and let Ap be the corresponding set of atomic propositions. Further, let $\mathbb{G}$ be the set of **G**-subformulas of φ, i.e. the subformulas of φ of the form $\mathbf{G}\psi$. We first describe the initial component without ϵ-transitions, then the accepting component, and then the ϵ-transitions linking the two.

5.1 Initial Component

As already sketched, the component keeps track in its state of the formula that must be satisfied by the rest of the word.

Definition 7. *The initial component for a formula φ is the transition system:*

$$\mathcal{N} = (2^{Ap}, \mathit{Reach}(\varphi), \mathit{af}, \varphi)$$

5.2 Accepting Component

The accepting component for a subset $\mathcal{G} \subseteq \mathbb{G}$ of **G**-subformulas is the product of one auxiliary monitor and one **G**-monitor for each $\mathbf{G}\psi \in \mathcal{G}$. First, we show how to build a **G**-monitor $\mathcal{U}$ for a single **G**-free formula. Second, we construct a product $\mathcal{P}$ of these **G**-monitors.

G-Monitor. Let ψ be a **G**-free formula. We construct a deterministic Büchi automaton $\mathcal{U}$, called the **G**-*monitor for* $\mathbf{G}\psi$, recognising $\mathsf{L}(\mathbf{G}\psi)$. We first give the definition, and then explain the intuition behind it.

Definition 8 (G-Monitor). *Let ψ be a **G**-free formula. The* **G**-monitor for ψ *is the deterministic Büchi automaton*

$$\mathcal{U}(\mathbf{G}\psi) = (2^{Ap}, \mathit{Reach}(\psi) \times \mathit{Reach}(\psi), \delta, (\psi, \mathbf{tt}), F)$$

where

- $\delta((\xi, \zeta), \nu) = \begin{cases} (\mathit{af}(\zeta, \nu) \wedge \psi, \mathbf{tt}) & \text{if } \mathit{af}(\xi, \nu) \equiv_P \mathbf{tt} \\ (\mathit{af}(\xi, \nu), \mathit{af}(\zeta, \nu) \wedge \psi) & \text{otherwise} \end{cases}$
- $F = \{((\xi, \zeta), \nu, p) \in Q \times 2^{Ap} \times Q \mid \mathit{af}(\xi, \nu) \equiv_P \mathbf{tt}\}$

The states of the monitor are pairs (ϕ_1, ϕ_2) of formulas. Intuitively, from state (ϕ_1, ϕ_2) the automaton checks if the rest of the run satisfies $\phi_1 \wedge \phi_2$, but starting with ϕ_1 and putting ϕ_2 “on hold”. The initial state is $(\psi, \mathbf{tt})$. After reading a letter, say ν_1, the automaton moves to $\delta(\psi, \mathbf{tt}) = (\mathit{af}(\psi, \nu_1), \psi)$. The meaning is: the automaton checks if the rest of the run satisfies $\mathit{af}(\psi, \nu_1) \wedge \psi$, but putting the check of ψ "on hold". If the next letter is, say, ν_2, then the automaton moves to $(af(\psi, \nu_1\nu_2), af(\psi, \nu_2) \wedge \psi)$, keeping the check of $af(\psi, \nu_2) \wedge \psi$ on hold. However, if $af(\psi, \nu_1\nu_2) = \mathbf{tt}$, then, the automaton knows already that the word $\nu_1\nu_2 \ldots$ satisfies ψ, the first check is complete, and the automaton "transfers" the checks kept on hold to the first position, moving to the state $(af(\psi, \nu_2) \wedge \psi, \mathbf{tt})$. The accepting transitions are those at which the automaton completes a check. If the automaton completes infinitely many checks, then all suffixes of the run satisfy ψ, and so the run satisfies $\mathbf{G}\psi$.

Lemma 4. *Let ψ be a $\mathbf{G}$-free formula and let w be a word, then $w \models \mathbf{G}\psi$ iff $\mathcal{U}(\mathbf{G}\psi)$ accepts w.*

Proof. Assume $w \models \mathbf{G}\psi$. Hence $\forall i.\, w_i \models \psi$. Since ψ is a $\mathbf{G}$-free formula, af will eventually derive $\mathbf{tt}$ due to Lemma 2, that is, $\forall i.\, \exists j.\, af(\psi, w_{ij}) \equiv_P \mathbf{tt}$. Hence $\mathcal{U}(\mathbf{G}\psi)$ visits infinitely many states (ξ, ζ) such that $\xi \equiv_P \mathbf{tt}$, and so it accepts.

Assume $w \not\models \mathbf{G}\psi$. Let i be a point where w fails to satisfy ψ ($w_i \not\models \psi$). Thus $af(\psi, w_{ij}) \not\equiv_P \mathbf{tt}$ for any j. Once $af(\psi, w_{ij})$ is propagated from the second component to the first, $\mathcal{U}(\mathbf{G}\psi)$ never uses an accepting transition again and hence does not accept. $\square$

Example 2. The $\mathbf{G}$-monitor for $\mathbf{G}\psi = \mathbf{G}(a \vee \mathbf{F}b)$ is the automaton of Figure 2.

$$\begin{tikzcd}
\arrow[r] & \boxed{p_0\colon (\psi, \mathbf{tt})} \arrow[loop right, Rightarrow, "a+b"] \arrow[d, bend right, "\bar{a}\bar{b}"'] \\
& \boxed{p_1\colon (\mathbf{F}b, \psi)} \arrow[u, bend right, Rightarrow, "b"'] \arrow[loop right, "a\bar{b}"] \arrow[d, "\bar{a}\bar{b}"] \\
& \boxed{p_2\colon (\mathbf{F}b, \mathbf{F}b)} \arrow[uu, bend left=60, Rightarrow, "b"] \arrow[loop right, "\bar{b}"]
\end{tikzcd}$$

Fig. 2. $\mathbf{G}$-monitor for $\mathbf{G}\psi = \mathbf{G}(a \vee \mathbf{F}b)$.

Product of G-Monitors. Let φ be a formula, and let $\mathbb{G}$ be the set of all $\mathbf{G}$-subformulas of φ. Fix a set $\mathcal{G} = \{\mathbf{G}\psi_1, \ldots, \mathbf{G}\psi_n\} \subseteq \mathbb{G}$. For every index $1 \leq i \leq n$, let $\mathcal{U}_i$ be the $\mathbf{G}$-monitor for the formula $\mathbf{G}(\psi_i[\mathcal{G}])$.

Definition 9 (Product of G-Monitors). *Let $\mathcal{U}_1, \ldots, \mathcal{U}_n$ as above, and let $\mathcal{U}_i = (2^{Ap}, Q_i, \delta_i, q_{0i}, F_i)$. The* product of $\mathbf{G}$-monitors *of φ with respect to $\mathcal{G}$ is the generalized deterministic Büchi automaton*

$$\mathcal{P}(\mathcal{G}) = (\ 2^{Ap},\ \prod_{i=1}^{n} Q_i, \prod_{i=1}^{n} \delta_i,\ (q_{01}, \ldots, q_{0n}),\ \{F_1', \ldots, F_n'\}\ )$$

where $(\,(q_1,\ldots,q_n),\nu,(q'_1,\ldots,q'_n)\,) \in F'_i$ *iff* $(q_i,\nu,q'_i) \in F_i$.

Lemma 5. *Let* $\mathcal{G}$ *be a set of* $\mathbf{G}$*-formulas and let* w *be a word. We have:* $\mathcal{P}(\mathcal{G})$ *accepts* w *iff* $w \models \mathbf{G}(\psi[\mathcal{G}])$ *for all* $\mathbf{G}\psi \in \mathcal{G}$.

Proof. Let $\mathcal{G} = \{\mathbf{G}\psi_1,\ldots,\mathbf{G}\psi_n\}$, and assume that the formulas are ordered so that if $\mathbf{G}\psi_i$ is a subformula of ψ_j then $j < i$ (so, in particular, $\mathbf{G}\psi_n$ is not a subformula of any of $\psi_1,\ldots,\psi_{n-1}$).

Assume $\mathcal{P}(\mathcal{G})$ accepts w. We prove $w \models \mathbf{G}(\psi_i[\mathcal{G}])$ for every $1 \le i \le n$ by induction on $n = |\mathcal{G}|$. If $n = 0$, then $\mathcal{P}(\emptyset)$ trivially accepts all words, and we are done. Let now $n > 0$. Since $\mathbf{G}\psi_n$ is not a subformula of any of $\psi_1,\ldots,\psi_{n-1}$, taking $\mathcal{G}' := \mathcal{G} \setminus \{\mathbf{G}\psi_n\}$ we have $\psi_i[\mathcal{G}] = \psi_i[\mathcal{G}']$ for every $\psi_i \in \mathcal{G}$. In particular, $\mathcal{P}(\mathcal{G})$ accepts w, then both $\mathcal{P}(\mathcal{G}')$ and $\mathcal{U}(\mathbf{G}(\psi_n[\mathcal{G}']))$ accept w too. By induction hypothesis we have $w \models \mathbf{G}(\psi_i[\mathcal{G}'])$ for every $1 \le i \le n-1$. By Lemma 4 we obtain $w \models \mathbf{G}(\psi_n[\mathcal{G}'])$. Finally, since $\psi_i[\mathcal{G}] = \psi_i[\mathcal{G}']$ for every $\psi_i \in \mathcal{G}$, we get $w \models \mathbf{G}(\psi_i[\mathcal{G}])$ for every $1 \le i \le n$.

The other direction is analogous. $\square$

5.3 Connecting the Initial and Accepting Components

We now define the ϵ-transitions leading from states in the initial component to states in the accepting component. Intuitively, each ϵ-transition corresponds to a guess $\mathcal{G}$, and starts the corresponding product $\mathcal{P}(\mathcal{G})$. However, recall that if the source of the transition is the state φ', then the accepting component must also check that the formula $\varphi'[\mathcal{G}])$ holds. For example, for the state q_2 of Figure 1 we have $\varphi' = \mathbf{G}\psi \wedge \mathbf{F}b$, and choosing $\mathcal{G} = \mathbf{G}\psi$ we get $\varphi'[\mathcal{G}] \equiv_P \mathbf{F}b$. So, intuitively, the state p_0 starts not only $\mathcal{P}(\mathcal{G})$, but also a deterministic automaton for $\mathbf{F}b$. More formally, p_0 is the initial state of the product of $\mathcal{P}(\mathcal{G})$ and this deterministic automaton.

The deterministic automaton for $\varphi'' := \varphi'[\mathcal{G}]$ is very simple. Since φ'' contains no occurrences of $\mathbf{G}$, by Lemma 2 we can just take the automaton $(2^{Ap}, \mathit{Reach}(\varphi''), \mathit{af}, \varphi'', \{\mathbf{tt}\})$. That is, the automaton keeps tracking the formula that the rest of the run must satisfy, and accepts iff eventually the formula is $\mathbf{tt}$. Comparing with the initial component $\mathcal{N}$, we observe that this automaton is nothing but $\mathcal{N}$ with $\mathbf{tt}$ as single accepting state, meaning all outgoing transitions are accepting. We can now define the complete limit-deterministic automaton for a formula φ.

Definition 10. *Let* φ *be a formula. Let* $\mathcal{N} = (2^{Ap}, Q_{\mathcal{N}}, \delta_{\mathcal{N}}, q_{0\mathcal{N}})$ *be the transition system of Definition 7. Furthermore, for every set* $\mathcal{G}$ *of* $\mathbf{G}$*-subformulas of* φ*, let* $\mathcal{P}(\mathcal{G}) = (2^{Ap}, Q_{\mathcal{P}(\mathcal{G})}, \delta_{\mathcal{P}(\mathcal{G})}, q_{0\mathcal{P}(\mathcal{G})}, \{F_{(\mathcal{G},\mathbf{G}\psi)} \mid \mathbf{G}\psi \in \mathcal{G}\})$ *be the product of Definition 9. The limit-deterministic automaton* $\mathcal{A}$ *is defined as*

$$\mathcal{A} = (2^{Ap},\ Q_{\mathcal{N}} \cup Q_{Acc},\ \delta_{\mathcal{N}} \cup \Delta_{\epsilon} \cup \Delta_{Acc},\ \varphi,\ \{F_{\xi} \mid \xi \in \mathbb{G}\})$$

where

$$Q_{Acc} = \bigcup_{\mathcal{G} \subseteq \mathbb{G}} (Q_{\mathcal{N}} \times Q_{\mathcal{P}(\mathcal{G})}) \quad \textit{and} \quad \Delta_{Acc} = \bigcup_{\mathcal{G} \subseteq \mathbb{G}} (\delta_{\mathcal{N}} \times \delta_{\mathcal{P}(\mathcal{G})})$$
$$\Delta_\epsilon = \{(\chi, \epsilon,\ (\chi[\mathcal{G}], q_{0\mathcal{G}})) \mid \chi \in Q_{\mathcal{N}}, \mathcal{G} \subseteq \mathbb{G}\}$$
$$F_\xi = \bigcup_{\mathcal{G} \subseteq \mathbb{G}} \left\{((\mathbf{tt}, q), \nu, (\mathbf{tt}, q')) \mid (q, \nu, q') \in F_{(\mathcal{G},\xi)}\right\}$$

Example 3. The complete automaton $\mathcal{A}$ for $\varphi = c \vee \mathbf{XG}(a \vee \mathbf{F}b)$, displayed in Figure 1, consists of the initial component above the dashed line, and the products of the auxiliary monitor with the product automata $\mathcal{P}$ for $\mathcal{G}_1 = \{\}$ and $\mathcal{G}_2 = \{\mathbf{G}\psi\}$ (below the dashed line)

5.4 Correctness

We prove a stronger correctness statement: For every state $\psi \in Q_{\mathcal{N}}$ of the initial component, the words w accepted from ψ are exactly those that satisfy ψ. Before we proceed, we need the notion of a *stable set* of $\mathbf{G}$-subformulas.

Definition 11. *A set $\mathcal{G} \subseteq \mathbb{G}$ of $\mathbf{G}$-subformulas of φ is stable for a word w if the following holds:*

$$\forall \mathbf{G}\psi \in \mathcal{G}.\ \forall i \geq 0.\ w_i \models \mathbf{G}\psi$$
$$\forall \mathbf{G}\psi \in \mathbb{G} \setminus \mathcal{G}.\ \forall i \geq 0.\ w_i \not\models \mathbf{G}\psi$$

Observe that at most one set is stable for a given word, and some words have no stable set. Further, if a set is stable for a word, then it is stable for all its suffixes.

Lemma 6. *Let φ be a formula, and let w be a word with stable set $\mathcal{G}$. Then $w \models \varphi$ iff $w \models \varphi[\mathcal{G}]$.*

Proof. (Induction on φ) The only non-trivial case is $\varphi = \mathbf{G}\varphi'$. Consider two subcases. If $\mathbf{G}\varphi' \in \mathcal{G}$, then by the definition of stable set we have $w \models \varphi$ and by the definition of $\varphi[\mathcal{G}]$ we get $\varphi[\mathcal{G}] = \mathbf{tt}$, so $w \models \varphi[\mathcal{G}]$. If $\mathbf{G}\varphi' \notin \mathcal{G}$, then $w \not\models \mathbf{G}\varphi'$ and $\varphi[\mathcal{G}] = \mathbf{ff}$, so $w \not\models \varphi[\mathcal{G}]$. ◻

Lemma 7. *Let φ be a formula, and let w be a word with stable set $\mathcal{G}$. If $w \models \varphi$ holds, then $\mathcal{A}$ has an accepting run r for w starting at φ. Moroever, r immediately switches from $\varphi \in Q_{\mathcal{N}}$ to the accepting component for $\mathcal{G}$.*

Proof. Assume $w \models \varphi$ and assume $\mathcal{G}$ is the stable set for w. Let r be the run starting at φ that immediately uses the ϵ-transition to jump to the accepting component for $\mathcal{G}$. We show that r is accepting. Since $\mathcal{G}$ is a stable set for w, the product automaton $\mathcal{P}(\mathcal{G})$ accepts w due to Lemma 5. It remains to prove that the auxiliary monitor eventually reaches $\mathbf{tt}$. By Lemma 6 we have $w \models \varphi[\mathcal{G}]$. Since $\varphi[\mathcal{G}]$ is $\mathbf{G}$-free, we can apply Lemma 2 and obtain that the auxiliary monitor eventually reaches $\mathbf{tt}$. ◻

Lemma 8. *Let w be a word and let $\varphi \in Q_{\mathcal{N}}$ be a state in the initial component of $\mathcal{A}$. We have: $w \models \varphi$ iff $\mathcal{A}$ has an accepting run r for w starting at φ.*

Proof. Assume $w \models \varphi$. Let $\mathcal{G} = \{\mathbf{G}\psi \in \mathbb{G} \mid w \models \mathbf{FG}\psi\}$ be the set of eventually true $\mathbf{G}$-subformulas, and let i be an index such that $w_i \models \mathbf{G}\psi$ for every $\mathbf{G}\psi \in \mathcal{G}$. Then $\mathcal{G}$ is a stable set for w_i. Consider the run that first reads $w_{0(i-1)}$ in the initial component, reaching a state φ', and then jumps to the accepting component for the $\mathcal{G}$. By Lemma 1 we have $w_i \models \varphi'$, and by Lemma 7 the run is accepting.

Assume $\mathcal{A}$ accepts w. Let r be an accepting run. Let i be the point at which r switches to the accepting component for some set $\mathcal{G}$. By Lemma 5 we have $w_i \models \xi$ for each $\xi \in \mathcal{G}$. Since accepting transitions are only taken when the remaining obligations are fulfilled (that is, when the formula in the first position of the tuple is replaced by $\mathbf{tt}$), we also obtain from Lemma 1 $w_i \models \mathit{af}(\varphi, w_{0(i-1)})[\mathcal{G}]$. Because the formulas are in NNF and $\mathcal{G} \subseteq \{\xi \in \mathbb{G} \mid w_i \models \xi\}$, we have $w_i \models \mathit{af}(\varphi, w_{0(i-1)})$, and, by Lemma 1, we finally get $w \models \varphi$. □

From this lemma we immediately obtain the correctness:

Theorem 1. *Let w be a word, then $w \models \varphi$ iff $\mathcal{A}$ accepts w.*

Further, staying for an arbitrary number of steps in the initial component is safe, because is it always possible to switch to a successful accepting component:

Lemma 9. *Let w be a finite word and let φ be a formula. We denote by $\mathsf{L}(q)$ the language accepted from state $q \in Q_{\mathcal{A}}$. Then the following inclusion holds:*

$$\bigcup_{q \in \delta(\varphi, w)} \mathsf{L}(q) \subseteq \mathsf{L}(\delta_{\mathcal{N}}(\varphi, w))$$

Proof. Let $q \in \delta(\varphi, w)$ be an arbitrary state in the accepting component reached after reading w. Let $w' \in \mathsf{L}(q)$ and let r' be an accepting run for w'. We extend r' to a run r for ww' starting in $\varphi \in Q_{\mathcal{N}}$ by taking the path from φ to q as a prefix for r'. Iteratively applying Lemma 1 and 8 yields $w' \in \mathsf{L}(\delta_{\mathcal{N}}(\varphi, w))$:

$$ww' \in \mathsf{L}(\varphi) \;\; \textit{iff} \;\; ww' \models \varphi \;\; \textit{iff} \;\; w' \models \mathit{af}(\varphi, w) \;\; \textit{iff} \;\; w' \in \mathsf{L}(\delta_{\mathcal{N}}(\varphi, w))$$

Thus $\mathsf{L}(q) \subseteq \mathsf{L}(\delta_{\mathcal{N}}(\varphi, w))$ for all $q \in \delta(\varphi, w)$. □

6 Optimisations

For the experimental evaluation we apply several optimisations to the presented construction:

First, *ϵ-transitions are removed* and replaced by the outgoing edges of the successor state. Second, *non-accepting sinks are removed*, as they can be easily recognised: from the state $\mathbf{ff}$ accepting edges are unreachable. Third, not all "ϵ-jumps" are necessary: A run starting in $\mathbf{G}\psi_1 \wedge \mathbf{G}\psi_2$ cannot be accepting in the component for $\emptyset$ or $\{\mathbf{G}\psi_1\}$, since the auxiliary monitor cannot reach $\mathbf{tt}$ from $\mathbf{ff}$ or $\mathbf{G}\psi_1 \wedge \chi$. Hence only jumps for (minimal) satisfying assignments of a state label (restricted to **G**s) are constructed. Fourth, we call a state *transient* if every run can only visit the state at most once, e.g. it is labelled by $\mathbf{X}\varphi$. As shown in Lemma 9 a jump to the accepting component can safely be delayed. Thus jumps for transient states are not constructed.

While these four optimisations suppress the construction of states and edges, also the state labels can be optimised: Fifth, **G**-monitors can safely replace (ξ, ζ) by $(\xi, \mathbf{tt})$ if ξ implies ζ. In a similar way the auxiliary component removes terms already taken care of by **G**-monitors. Sixth, all modal operators in state labels are unfolded, reducing the number of states: $\mathbf{FG}a$ is rewritten to $(\mathbf{FG}a) \vee (\mathbf{G}a)$ and merged with existing states.

7 Complexity

Upper Bound. Let n be the length of the formula φ. Since $Q_{\mathcal{U}(\mathbf{G}\psi)}$ and $Q_{\mathcal{N}}$ are defined using *Reach*, the size for $Q_{\mathcal{U}(\mathbf{G}\psi)}$ and $Q_{\mathcal{N}}$ is $\mathcal{O}(2^{2^n})$. For each $\mathcal{G} \subseteq \mathbb{G}$ the accepting component has at most $|\mathbb{G}|$ **G**-monitors and one auxiliary monitor. Hence the size of the accepting component for a single $\mathcal{G}$ is at most $(|\mathbb{G}| + 1) \cdot \mathcal{O}(2^{2^n}) = \mathcal{O}(2^{2^{n+\log\log n}})$. Summing up to:

$$\mathcal{O}(2^{2^n}) + 2^{|\mathbb{G}|} \cdot \mathcal{O}(2^{2^{n+\log\log n}}) = \mathcal{O}(2^{2^{n+\log n+\log\log n}})$$

Lower Bound. This upper bound is matched by a double exponential (not tight) lower bound. The language used in the proof is an adaptation of [21].

Theorem 2. *There is a family of formulas ϕ_n of size $\mathcal{O}(n^2)$ such that the smallest limit-deterministic state-based Büchi automaton recognising $\mathsf{L}(\phi_n)$ has at least 2^{2^n} states.*

Proof. For every $n \in \mathbb{N}$, let r_n be the regular expression over the alphabet $\Sigma = \{0, 1, \#, \$, \%\}$, defined as

$$r_n := \sum_{w \in \{0,1\}^n} \% \ (0|1|\#)^* \ \# \ w \ \# \ (0|1|\#)^* \ \$ \ w$$

Since the language $\mathsf{L}(r_n^\omega)$ can be expressed by an LTL formula of size $\mathcal{O}(n^2)$ (Lemma 10) and the smallest limit-deterministic state-based Büchi automaton recognising $\mathsf{L}(r_n^\omega)$ has at least 2^{2^n} states (Lemma 11), the claim holds. □

Lemma 10. *There exists an LTL formula of size $\mathcal{O}(n^2)$ defining $\mathsf{L}(r_n^\omega)$.*

Proof. The conjunction of the following LTL formulas of size $\mathcal{O}(n^2)$ defines $\mathsf{L}(r_n^\omega)$:

$$\% \wedge \mathbf{G}(\$ \rightarrow \mathbf{X}^{n+1}\%) \wedge \mathbf{G}(\bigvee_{\alpha \in \Sigma} (\alpha \wedge \bigwedge_{\beta \in \Sigma \setminus \{\alpha\}} \neg\beta)) \tag{1}$$

$$\mathbf{G}(\% \rightarrow \mathbf{X}((0 \vee 1 \vee \#)\, \mathbf{U}(\bigwedge_{1 \leq i \leq n} (\varphi(i,0) \vee \varphi(i,1)) \wedge \mathbf{X}^{n+1}\#))) \tag{2}$$

with $\varphi(i, \alpha) := \mathbf{X}^i \alpha \wedge ((0 \vee 1 \vee \#) \, \mathbf{U}(\$ \wedge \mathbf{X}^i \alpha))$. Formula 1 ensures the basic syntactic properties of r_n, and formula 2 enforces that the literal % is always succeeded by a string of the form $(0|1|\#)^*\#w\#(0|1|\#)^*\$w$ using $\varphi(i, \alpha)$ to guarantee the existence of matching substrings $\#w\#$ and $\$w$. □

Lemma 11. *The smallest limit-deterministic state-based Büchi automaton recognising $\mathsf{L}(r_n^\omega)$ has at least 2^{2^n} states.*

Proof. Let $\mathcal{A}_n$ be a limit-deterministic state-based Büchi automaton recognising $\mathsf{L}(r_n^\omega)$. A state q is called a *\$-successor* if there exists an accepting run r containing the transition $(p, \$, q)$ and q is in the deterministic component of $\mathcal{A}_n$. For every \$-successor q we denote by $\mathcal{W}(q) = \{u \in \{0,1\}^n \mid \exists w.\ uw \in \mathsf{L}(q)\}$ the set of prefixes of length n of $\mathsf{L}(q)$. We show that $\mathcal{A}_n$ has at least one distinct \$-successor q for each subset $\emptyset \neq \mathcal{S} \subseteq \{0,1\}^n$, such that $\mathcal{W}(q) = \mathcal{S}$. Thus $\mathcal{A}_n$ has at least 2^{2^n}states, since not all states can be \$-successors. We expose these states by the recursively defined sequence s. Let $\mathcal{S} = \{w_1, w_2, \ldots w_i\}$ be such a subset and let $k := |Q_n|$:

$$
\begin{aligned}
s(\mathcal{S}) &:= s_{|\mathcal{S}|}(\mathcal{S})\\
s_1(\mathcal{S}) &:= \%\#w_1\#\ldots\#w_i\#\$w_1\\
s_j(\mathcal{S}) &:= s_{j-1}(\mathcal{S})^k \cdot s_{j-2}(\mathcal{S})^k \cdot \ldots \cdot s_1(\mathcal{S})^k \,\%\#w_1\# \cdot \ldots \cdot \#w_i\#\$w_j \ \forall 1 < j \leq i
\end{aligned}
$$

Since $s(\mathcal{S})^\omega \in L(r_n^\omega)$ holds for all non-empty $\mathcal{S}$, there exists for each $s(\mathcal{S})$ an accepting run r, such that the sequence is read in the deterministic component. Furthermore by construction we have for each \$-successor q encountered reading $s(\mathcal{S})$: $\mathcal{W}(q) \subseteq \mathcal{S}$. Showing that the converse also holds concludes the proof:

Proposition 1. *Let $\mathcal{S} = \{w_1, \ldots w_i\}$, and let $1 \leq j \leq i$. Furthermore, assume the sequence $s_j(\mathcal{S})$ is read in the deterministic part during an accepting run. Let q_j be the \$-successor transitioned to after the last \$ of $s_j(\mathcal{S})$. Then q_j satisfies $\mathcal{W}(q_j) \supseteq \{w_1, \ldots, w_j\}$.*

Proof (By induction on j). Case $j = 1$ is trivial. Case $j > 1$: By construction $s_j(\mathcal{S})$ is equal to $s_{j-1}(\mathcal{S})^{k+1}$ up to the last \$ and ends with w_j instead of w_{j-1}. Since $k = |Q_n|$ and the sequence is read in the deterministic component, q_j occurred previously in the run on $s_{j-1}(\mathcal{S})$ as a \$-successor. Hence we apply the induction hypothesis to q_j and obtain: $\mathcal{W}(q_j) \supseteq \{w_1, \ldots, w_{j-1}\}$. Since w_j occurs after the last \$-symbol in $s_j(\mathcal{S})$, we have $\mathcal{W}(q_j) \supseteq \{w_1, \ldots, w_j\}$. □

8 Quantitative Probabilistic Model Checking

The problem of LTL probabilistic model checking [3] is to determine the probability that an LTL formula φ holds on a run generated by a given Markov chain $\mathcal{M}$, i.e., $\mathbb{P}_\mathcal{M}\{\text{run } \rho \mid \rho \models \varphi\}$, or more generally, for Markov decision process $\mathcal{M}$ the *maximal* probability that φ is satisfied, i.e., $\sup_\sigma \mathbb{P}_{\mathcal{M}^\sigma}\{\text{run } \rho \mid \rho \models \varphi\}$, where σ ranges over *schedulers* resolving the non-determinism of $\mathcal{M}$, and $\mathcal{M}^\sigma$ is the Markov chain resulting from application of σ to $\mathcal{M}$.

The automata-theoretic approach to model checking LTL over Markov decision processes amounts to (1) constructing the product $\mathcal{M} \times \mathcal{A}$ of the system $\mathcal{M}$ and an automaton $\mathcal{A}$ for the LTL formula, (2) computing *maximal end components (MECs)* in the product, (3) determining which MECs are accepting, and (4) determining the maximal probability to reach the accepting MECs.

However, as opposed to the non-probabilistic model checking case, in general the automaton $\mathcal{A}$ cannot be used if it is non-deterministic. Intuitively, resolving non-determinism of the automaton may depend on the yet unknown, probabilistically given future. For the same reason, limit-deterministic automata are in general applicable only to qualitative probabilistic model checking, i.e., determining whether the satisfaction probability is 0, 1, or neither. We show that our limit-deterministic automata can be used even in the quantitative-analysis algorithm outlined above without conversion to a fully deterministic automaton.

Notice that the only non-deterministic transitions present in our construction are ϵ-transitions. This allows us to represent the non-deterministic choice in our LDBA $\mathcal{A}$ by means of additional ϵ-actions in the product MDP where each ϵ-action only changes the automaton state. Formally, given a Markov decision process $\mathcal{M} = (S, \mathrm{Act}, \mathbf{P}, s_0, \mathrm{Ap}, L)$ with set of states S, set of actions Act, transition probability function $\mathbf{P} : S \times \mathrm{Act} \times S \to [0,1]$, initial state $s_0 \in S$ and labelling function L, and given an automaton $\mathcal{A} = (2^{\mathrm{Ap}}, Q, \delta, q_0, Acc)$ we define the product $(\mathcal{M} \times \mathcal{A}) = \big((S \times Q), \mathrm{Act}', \mathbf{P}', (s_0, q_0), \mathrm{Ap}, L'\big)$ of $\mathcal{M}$ and $\mathcal{A}$ as follows. Firstly, for every potential ϵ-transition in $\mathcal{A}$ to some state $q \in Q$ we add a corresponding action ϵ_q in the product:

$$\mathrm{Act}' := \mathrm{Act} \cup \{\epsilon_q \mid q \in Q\}$$

As usual, we define for all $(s,q), (s',q') \in S \times Q$ and $\alpha \in \mathrm{Act}$:

$$L'\left((s,q)\right) := L(s)$$
$$\mathbf{P}'\left((s,q), \alpha, (s',q')\right) := \begin{cases} \mathbf{P}(s, \alpha, s') & \text{if } q \xrightarrow{L(s')} q' \\ 0 & \text{otherwise} \end{cases}$$

Additionally, for all $(s,q), (s',q') \in S \times Q$, the transition probabilities of the ϵ-actions are given by

$$\mathbf{P}'\left((s,q), \epsilon_{\hat{q}}, (s',q')\right) := \begin{cases} 1 & \text{if } s = s' \text{ and } q \xrightarrow{\epsilon} q' = \hat{q} \\ 0 & \text{otherwise} \end{cases}$$

Now we are going to show that our product can indeed be used for quantitative model checking using the standard techniques.

Theorem 3. *For any formula φ, the automaton $\mathcal{A}$ can be used in the standard probabilistic model checking algorithm, i.e., for any Markov decision process $\mathcal{M}$,*

$$\sup_\sigma \mathbb{P}_{\mathcal{M}^\sigma}[\mathsf{L}(\mathcal{A})] = \sup_\sigma \mathbb{P}_{(\mathcal{M} \times \mathcal{A})^\sigma}[\Diamond X]$$

where $\Diamond X$ is the set of runs that reach an accepting MEC of $\mathcal{M} \times \mathcal{A}$.

Proof. The inequality "$\geq$" is trivial even for general non-deterministic automata. Indeed, every scheduler over $\mathcal{M} \times \mathcal{A}$ induces a scheduler over $\mathcal{M}$ (by elimination of ϵ-transitions and subsequent projection) such that for every run (ρ, π) reaching X, where acceptance is guaranteed, we have a run ρ of $\mathcal{M}$ that is accepted by $\mathcal{A}$ due to π. The scheduler thus resolved the non-determinism also of the automaton.

While the inequality "$\leq$" generally holds only for deterministic automata, we prove it also for our $\mathcal{A}$. We define a random variable *index* mapping a run ρ of $\mathcal{M}$, corresponding to a word w, as follows: $\mathit{index}(\rho) := \min\{i \mid \forall \mathbf{G}\psi \in \mathbb{G}.\, w \models \mathbf{FG}\psi \implies w_i \models \mathbf{G}\psi\}$. Observe that every run has a finite index. From this point on, the run satisfies all $\mathbf{G}$-formulas that it will ever eventually satisfy; we call the set of these formulas $\mathcal{G}(\rho)$.

Given a scheduler σ, a state m of $\mathcal{M}^\sigma$ is called *decided* if almost all runs starting in m have index 0. In other words, almost all runs of $\mathcal{M}^\sigma$ starting in m satisfy each $\mathbf{G}$-formula already at the beginning or never. Intuitively, $\mathcal{G}$ is determined by m.

Lemma 12. *Let C be a bottom strongly connected component (BSCC) of a Markov chain. Then all states of C are decided and almost all runs ρ in C have the same $\mathcal{G}(\rho)$.*

Proof. Let $\mathbb{P}_s[\mathcal{L}]$ denote the probability that a run of the Markov chain starting in state s induces a word from language $\mathcal{L}$. Now let $\mathbf{G}\psi \in \mathbb{G}$. If for all $c \in C$, $\mathbb{P}_c[\mathsf{L}(\mathbf{G}\psi)] = 1$, then we are done. Let now $c \in C$ be such that $\mathbb{P}_c[\mathsf{L}(\mathbf{G}\psi)] < 1$. We show that for all $d \in C$, we have $\mathbb{P}_d[\mathsf{L}(\mathbf{G}\psi)] = 0$, thus also proving $\mathbb{P}_d[\mathsf{L}(\mathbf{FG}\psi)] = 0$. Since $\mathbb{P}_c[\mathsf{L}(\mathbf{G}\psi)] < 1$, we have $\mathbb{P}_c[\mathsf{L}(\mathbf{F}\neg\psi)] > 0$. Therefore, with every visit of c there is a positive probability p that ψ will be violated in the next n steps for some $n \in \mathbb{N}$. Since c is in a BSCC it will be visited infinitely often with probability 1 from any $d \in C$. Consequently, $\mathbb{P}_d[\mathsf{L}(\mathbf{G}\psi)] \leq \lim_{k\to\infty}(1-p)^k = 0$. $\square$

We are now ready to prove the inequality "$\leq$". Given a scheduler σ of $\mathcal{M}$, we define a scheduler σ' of $\mathcal{M} \times \mathcal{A}$ such that $\mathbb{P}_{\mathcal{M}^\sigma}[\mathsf{L}(\mathcal{A})] \leq \mathbb{P}_{(\mathcal{M}\times\mathcal{A})^{\sigma'}}[\Diamond X]$. The scheduler σ' follows the behaviour of σ up to the point where a BSCC is reached in $\mathcal{M}^\sigma$. A run on $\mathcal{M}^\sigma$ will almost surely reach some BSCC in $\mathcal{M}^\sigma$. Let m be the first visited state in a BSCC of $\mathcal{M}^\sigma$. By Lemma 12, m has unique decided $\mathcal{G}$, and σ' then chooses the unique ϵ-action ϵ_q such that $q \in Q_\mathcal{N} \times Q_{\mathcal{P}(\mathcal{G})}$. Having performed this ϵ-action, the scheduler σ' then continues to follow the behaviour of σ indefinitely. Note that apart from emulating σ, the constructed scheduler σ' only decides when to switch to the accepting component and with which $\mathcal{G}$.

Notice that by construction every run ρ_σ on $\mathcal{M}^\sigma$ corresponds to a run $\rho_{\sigma'}$ on $(\mathcal{M} \times \mathcal{A})^{\sigma'}$ with equal transition probabilities, except for the probability of the ϵ-action, which has neutral probability 1. It thus only remains to show that if a trace of ρ_σ is accepted by $\mathcal{A}$ then the corresponding run $\rho_{\sigma'}$, projected to the second component, is also an accepting run on $\mathcal{A}$. To this end, let *All* be the set of states where $\mathcal{A}$ can be after reading the run up to the point where m is reached. In particular, let $\mathit{init}, \mathit{acc} \in \mathit{All}$ be the source and the target of the ϵ-transition taken if σ' is followed, i.e. when the transition from (m, init) to (m, acc) under action ϵ_{acc} is taken. Note that other elements of *All* correspond to runs of $\mathcal{A}$ that either switch at another time or to an accepting part $Q_\mathcal{N} \times Q_{\mathcal{P}(\mathcal{G}')}$

for a different $\mathcal{G}'$. In order to show that σ' always chooses an accepting run of $\mathcal{A}$ if there is one, we have to show that the following constraints are satisfied:

– Lingering in the initial part and delaying the switch to the acceptance part is safe. Formally:

$$\mathsf{L}(\mathit{init}) \supseteq \bigcup_{a \in All} \mathsf{L}(a) \tag{1}$$

This is shown in Lemma 9.

– Switching to the acceptance part is safe upon reaching the BSCC. Formally:

$$\mathbb{P}_m[\mathsf{L}(\mathit{init})] = \mathbb{P}_m[\mathsf{L}(\mathit{acc})] \tag{2}$$

From Lemma 12 we obtain a unique $\mathcal{G}$ for almost all runs starting in m. Let $\mathsf{L}(\mathcal{G})$ denote the set of runs satisfying $\mathbf{FG}\psi$ for all $\mathbf{G}\psi \in \mathcal{G}$ and not satisfying $\mathbf{FG}\psi$ for all $\mathbf{G}\psi \notin \mathcal{G}$. Further, the set of runs not in $\mathsf{L}(\mathcal{G})$ has zero probability, formally $\mathbb{P}_m[\overline{\mathsf{L}(\mathcal{G})}] = 0$. Hence it is sufficient to show $\mathsf{L}(\mathit{init}) \cap \mathsf{L}(\mathcal{G}) = \mathsf{L}(\mathit{acc}) \cap \mathsf{L}(\mathcal{G})$. Here $\supseteq$ is trivial and for $\subseteq$ let $w \in \mathsf{L}(\mathit{init}) \cap \mathsf{L}(\mathcal{G})$. Observe that $\mathcal{G}$ is a stable set for w. Let φ_{init} be the formula label of init. Because $w \in \mathsf{L}(\mathit{init})$, $w \models \varphi_{\mathit{init}}$. Thus we can apply Lemma 7 and obtain $w \in \mathsf{L}(\mathit{acc})$. $\square$

Remark 2. The limit-deterministic automata of [6] (making its first part deterministic, which is informally mentioned as an option in the paper) and the one of [14] (which is essentially the same) satisfy condition (1). Moreover, they satisfy condition (2), not necessarily upon reaching the BSCC of $\mathcal{M}^\sigma$, but at the latest upon reaching the BSCC of its product with the initial part of the automaton, see [14]. So they can also be used for quantitative probabilistic model checking using the standard algorithm, as implicitly suggested by the more complex on-the-fly variant of the algorithm of [14].

9 Experimental Evaluation

In our experimental evaluation we measure the state size and the number of acceptance sets. We compare our translation to state-of-the-art LTL-to-Büchi and LTL-to-deterministic-Rabin translators.

For the first group we use the semi-determinisation of [6] to obtain limit-deterministic systems where the first component only uses the non-determinism for jumping into the second component, starting a breakpoint construction. This approach enables quantitative model checking (see Remark 2). The algorithm is only applied when the automaton is non-deterministic and this translation is necessary. Apart from selecting state-based Büchi automata as output, the tools are left in their default configuration.

We compared the following tools[2] where each tool is given at most 4GB of memory and 5 minutes computing time:

[2] `ltl2ba` was left out and the improved "successor" `ltl3ba` was used.

L3 (`ltl3ba`, 1.1.2) - An enhanced fork of `ltl2ba` with formula rewriting. [1]

S (`ltl2tgba`, 1.99.6) - The LTL translator from Spot. Features advanced formula simplification and several post-processing optimisations. [9]

L2D (`ltl2dstar`, 0.5.3) - Translates LTL to deterministic Streett and Rabin automata. Configured to use Spot as a translator from LTL to NBA. [17]

R (`Rabinizer`, 3.1) - Constructs deterministic generalized Rabin automata avoiding Safra's construction. Configured to produce Rabin automata. [19]

LD (`ltl2ldba`) - Implementation of the proposed construction without any post-processing. Available at: `www7.in.tum.de/~sickert/projects/ltl2ldba`

	Büchi Acceptance			Rabin Acceptance	
	L3	S	LD	L2D	R
$j = 1$	10 (1)	11 (1)	3 (1)	5 (4)	2 (4)
$j = 2$	21 (1)	21 (1)	4 (2)	17 (6)	3 (6)
$j = 3$	44 (1)	44 (1)	5 (3)	49 (8)	4 (8)
$j = 4$	95 (1)	95 (1)	6 (4)	129 (10)	5 (10)
$k = 2$	40 (1)	62 (1)	5 (2)	4385 (14)	13 (8)
$k = 3$	326 (1)	571 (1)	9 (3)	*	198 (16)
φ_1	37 (1)	12 (1)	9 (3)	6 (4)	7 (6)
φ_2	39 (1)	33 (1)	7 (3)	27 (4)	6 (6)
φ_3	35 (1)	14 (1)	19 (3)	7 (6)	13 (6)
$f(0,0)$	5 (1)	5 (1)	5 (1)	5 (2)	5 (4)
$f(0,2)$	16 (1)	16 (1)	10 (1)	10 (4)	7 (4)
$f(0,4)$	18 (1)	18 (1)	16 (1)	12 (4)	9 (4)
$f(1,0)$	51 (1)	64 (1)	6 (3)	196 (10)	17 (6)
$f(1,2)$	138 (1)	345 (1)	28 (3)	109839 (22)	33 (6)
$f(1,4)$	943 (1)	2450 (1)	58 (3)	*	70 (6)
$f(2,0)$	289 (1)	215 (1)	10 (4)	99793 (22)	41 (8)
$f(2,2)$	915 (1)	1061 (1)	46 (4)	*	94 (8)
$f(2,4)$	12314 (1)	14161 (1)	92 (4)	*	139 (8)
Σ [5]	55444 (353)***	29559 (356)	9518 (889)**	382350 (1670)*	10616 (2554)**

Table 1. Number of states and number of acceptance sets in parenthesis for the constructed automata. The smallest and second smallest state spaces are highlighted. Each resource exhaustion is marked with an additional *. Abbreviated formulas: $\varphi_1 = \mathbf{GF}(\mathbf{F}a \vee \mathbf{G}b \vee \mathbf{FG}(a \vee (\mathbf{X}b))$, $\varphi_2 = \mathbf{FG}(\mathbf{G}a \vee \mathbf{F}\neg b \vee \mathbf{GF}(a \wedge \mathbf{X}b))$, $\varphi_3 = \mathbf{GF}(\mathbf{F}a \vee \mathbf{GX}b \vee \mathbf{FG}(a \vee \mathbf{XX}b))$

We consider the following five groups of formulas:

1. The first group of formulas in Table 1 are from the GR(1) fragment of LTL and parametrised by j: $\bigwedge_{i=1}^{j}(\mathbf{GF}a_i) \implies \bigwedge_{i=1}^{j}(\mathbf{GF}b_i)$. These have been previously used in [20].

2. The second group of formulas are fairness constraints, taken from [11], and are parametrised by k: $\bigwedge_{i=1}^{k}(\mathbf{GF}a_i \vee \mathbf{FG}b_i)$
3. The third section looks at formulas with light nesting of modal operators.
4. In the fourth section we explore the effect of deeply nesting modal operators using the parametrised formula f.

$$f(0,j) = (\mathbf{GF}a_0)\mathbf{U}(\mathbf{X}^j b) \qquad f(i+1,j) = (\mathbf{GF}a_{i+1})\mathbf{U}(\mathbf{G}f(i,j))$$

   While the automata sizes are close to each other for $i, j = 0$, this immediately changes after increasing the parameters controlling the nesting depth.
5. The last entry is cumulative and sums up results for 359 formulas. These formulas are from the collection used in [5]. The authors collected these from existing sources, such as [23, 10, 12, 13], and additionally included randomly generated formulas.

10 Conclusion

We present a direct translation from LTL formulas to limit-deterministic Büchi automata. The approach relies on decomposing the formula, constructing small automata for each $\mathbf{G}$ and building a product deciding acceptance. The complexity analysis shows that translating LTL to limit-deterministic automata is in the worst case double exponential, which is matched by existing constructions and the novel approach. The experimental section shows that for deeply nested formulas the presented approach outperforms existing tools and in the general case is as good as the other tools.

There are several open questions we want to investigate. First, we have not included the *Release*-operator in the syntax. While this does not have an impact on the expressiveness of the logic, direct support for it would be favourable. Second, it is open, if is possible to adapt the construction such that for the LTL$_{\setminus\mathbf{GU}}$ fragment the size is also exponential as in [16]. Third, we would like to investigate the impact of specialised and standard post-processing steps on the size of the automaton. Fourth, we want to study the performance impact of using this construction for quantitative model checking.

Finally, in the area of reactive synthesis DRAs can be replaced by *good for games* Parity automata [15, 18]. Studying the connection to our translation, and a possibility of adapting our work for reactive synthesis is open.

Acknowledgments. The authors want to thank Orna Kupferman for suggesting the language for proving the lower complexity bound.

References

1. Babiak, T., Křetínský, M., Rehák, V., Strejcek, J.: LTL to Büchi automata translation: Fast and more deterministic. In: TACAS. LNCS, vol. 7214, pp. 95–109 (2012)

2. Baier, C., Kiefer, S., Klein, J., Klüppelholz, S., Müller, D., Worrel, J.: Markov chains and unambiguous Büchi automata. In: *To appear in* CAV 2016, preprint available at `http://arxiv.org/abs/1605.00950` (2016)
3. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
4. Blahoudek, F., Heizmann, M., Schewe, S., Strejcek, J., Tsai, M.: Complementing semi-deterministic Büchi automata. In: TACAS. LNCS, vol. 9636, pp. 770–787 (2016)
5. Blahoudek, F., Křetínský, M., Strejček, J.: Comparison of LTL to deterministic Rabin automata translators. In: LPAR. LNCS, vol. 8312, pp. 164–172 (2013)
6. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM 42(4), 857–907 (1995)
7. Couvreur, J., Saheb, N., Sutre, G.: An optimal automata approach to LTL model checking of probabilistic systems. In: LPAR. LNCS, vol. 2850, pp. 361–375 (2003)
8. Duret-Lutz, A.: Manipulating LTL formulas using Spot 1.0. In: ATVA. LNCS, vol. 8172, pp. 442–445 (2013)
9. Duret-Lutz, A.: LTL translation improvements in Spot 1.0. International Journal on Critical Computer-Based Systems 5(1/2), 31–54 (Mar 2014)
10. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE. pp. 411–420 (1999)
11. Esparza, J., Křetínský, J.: From LTL to deterministic automata: A safraless compositional approach. In: CAV. LNCS, vol. 8559, pp. 192–208 (2014)
12. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: CAV. LNCS, vol. 2102, pp. 53–65 (2001)
13. Geldenhuys, J., Hansen, H.: Larger automata and less work for LTL model checking. In: SPIN. LNCS, vol. 3925, pp. 53–70 (2006)
14. Hahn, E.M., Li, G., Schewe, S., Turrini, A., Zhang, L.: Lazy probabilistic model checking without determinisation. In: CONCUR. LIPIcs, vol. 42, pp. 354–367 (2015)
15. Henzinger, T.A., Piterman, N.: Solving games without determinization. In: CSL. LNCS, vol. 4207, pp. 395–410. Springer (2006)
16. Kini, D., Viswanathan, M.: Limit deterministic and probabilistic automata for LTL \ GU. In: TACAS. LNCS, vol. 9035, pp. 628–642 (2015)
17. Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theor. Comput. Sci. 363(2), 182–195 (2006)
18. Klein, J., Müller, D., Baier, C., Klüppelholz, S.: Are good-for-games automata good for probabilistic model checking? In: LATA. LNCS, vol. 8370, pp. 453–465 (2014)
19. Komárková, Z., Křetínský, J.: Rabinizer 3: Safraless translation of LTL to small deterministic automata. In: ATVA. LNCS, vol. 8837, pp. 235–241 (2014)
20. Křetínský, J., Esparza, J.: Deterministic automata for the (F,G)-fragment of LTL. In: CAV. LNCS, vol. 7358, pp. 7–22 (2012)
21. Kupferman, O., Vardi, M.Y.: From linear time to branching time. ACM Trans. Comput. Log. 6(2), 273–294 (2005)
22. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. LNCS, vol. 6806, pp. 585–591 (2011)
23. Pelánek, R.: BEEM: benchmarks for explicit model checkers. In: SPIN. LNCS, vol. 4595, pp. 263–267 (2007)
24. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Logical Methods in Computer Science 3(3) (2007)
25. Safra, S.: On the complexity of omega-automata. In: FOCS. pp. 319–327 (1988)
26. Schewe, S.: Tighter bounds for the determinisation of Büchi automata. In: FoSSaCS. LNCS, vol. 5504, pp. 167–181 (2009)
27. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS. pp. 327–338 (1985)