Are Good-for-Games Automata Good for Probabilistic Model Checking?[⋆]

Extended Version – 09.04.2014

Joachim Klein, David Müller, Christel Baier, and Sascha Klüppelholz

Institute of Theoretical Computer Science
Technische Universität Dresden, 01062 Dresden, Germany
{klein,david.mueller,baier,klueppel}@tcs.inf.tu-dresden.de

Abstract. The potential double exponential blow-up for the generation of deterministic ω-automata for linear temporal logic formulas motivates research on weaker forms of determinism. One of these notions is the *good-for-games* property that has been introduced by Henzinger and Piterman together with an algorithm for generating good-for-games automata from nondeterministic Büchi automata. The contribution of our paper is twofold. First, we report on an implementation of this algorithms and exhaustive experiments. Second, we show how good-for-games automata can be used for the quantitative analysis of systems modeled by Markov decision processes against ω-regular specifications and evaluate this new method by a series of experiments.

1 Introduction

The automata-theoretic approach to formal verification relies on the effective translation of specifications, e.g., formulas of some temporal logic such as linear temporal logic (LTL) into automata over infinite words (ω-automata) [34,6,12]. The verification problem for finite-state system models is then solvable by analyzing the product of the system model and the automaton for the formula. In the classical setting where the system model can be seen as a nondeterministic automaton, *nondeterministic* ω-automata suffice. For some applications, such as game-based synthesis and probabilistic model-checking problems, the nondeterminism of the ω-automaton poses a problem. Used as a monitor to determine the winning strategies of turn-based two-player games, the lack of look-ahead beyond the players' choices in general precludes the use of nondeterministic automata. Similarly, in probabilistic model checking, the lack of look-ahead beyond the probabilistic choices renders nondeterministic automata unsuitable in general. In these settings, the use of *deterministic ω-automata* resolves these problems at the cost of a further worst-case exponential determinization construction [26,31,25]. Thus, there is considerable interest in methods that tackle the worst-case double exponential time-complexity of algorithms caused by the construction of deterministic ω-automata for LTL formulas. This includes for example variants of the

⋆ Funded by the DFG through the Graduiertenkolleg 1763 (QuantLA), the CRC 912 HAEC, the cluster of excellence cfAED and the project QuaOS and by the ESF young researcher group IMData 100098198 and the EU-FP-7 grant 295261 (MEALS).

determinization construction for nondeterministic Büchi automata (NBA) [28,32], heuristics [14,15] and the direct translation from fragments of LTL to deterministic automata [24,17,1]. Instead of reducing the number of states, [27] provides a translation from non-confluent NBA that aims to generate a compact symbolic representation of the generated deterministic automata based on binary decision diagrams (BDDs).

There are also several attempts to avoid determinization in certain scenarios [21,18] and provide better theoretical complexity and performance in practice. Henzinger and Piterman [13] introduce a special property for nondeterministic automata, being *good-for-games* (GFG), that is fulfilled by all deterministic automata but still permits nondeterministic choices. [13] proposes an algorithm, called the HP-algorithm here, for the construction of a nondeterministic GFG automaton with parity acceptance from an NBA that is amenable to a symbolic representation. The number of states in the constructed GFG automaton is still exponential in the number of states of the given NBA, but a smaller worst-case bound on the number of states can be provided than for Safra's determinization algorithm [31]. Among others, [4] introduced the notion *determinizable-by-pruning* for automata that have an embedded deterministic automaton for the same language. [4] states the existence of GFG automata that are not determinizable-by-pruning, but we are not aware of any result stating the existence of languages where GFG automata are more succinct than deterministic ones. To the best of our knowledge, the HP-algorithm is the sole published algorithm for the construction of GFG automata and it has not been implemented or experimentally evaluated yet.

In the context of probabilistic model checking for finite-state Markov chains, [8,3] propose the use of unambiguous automata that can be generated from LTL formulas with a single exponential blow-up in the worst case. Alternative approaches that also lead to single exponential-time model-checking algorithms for Markov chains and LTL specifications have been presented in [5] using weak alternating automata and in [7] using an iterative approach to integrate the effect of the temporal modalities of a given LTL formula φ in the Markov chain. Given that the analogous problem is 2EXPTIME-complete for models where nondeterministic and probabilistic choices alternate [7], there is no hope to generalize these results for Markov decision processes (MDPs). Only for the qualitative analysis of MDPs where the task is to show that an ω-regular path property holds with probability 1, no matter how the nondeterminism is resolved, Büchi automata that are deterministic-in-limit are shown to be sufficient [34,7].

Contribution. The purpose of our paper is to study whether GFG automata are adequate in the context of probabilistic model checking, both at the theoretical and the practical level. At the theoretical level, we answer in the affirmative and provide algorithms for the computation of maximal or minimal probabilities for path properties specified by GFG automata in finite-state Markov decision processes (MDPs). The time complexity of our algorithm is polynomial in the size of the given MDP and GFG automaton. To evaluate the GFG-based approach empirically, we have implemented the HP-algorithm (and various variants) symbolically using binary decision diagrams (BDDs). In a series of experiments, we study the performance of the HP-algorithm – from LTL formula via NBA to GFG automaton – compared to the determinization implementation of LTL2DSTAR [14,15] based on Safra's construction. We have furthermore implemented the GFG-based approach for the analysis of MDPs in the popular probabilistic model checker PRISM [22] and evaluated its performance in practice.

Outline. Section 2 briefly introduces our notations for ω-automata and MDPs. The applicability of GFG automata for the quantitative analysis of MDPs is shown in Section 3. In Section 4, we study the HP-algorithm in detail and present a few heuristics that have been integrated in our implementation. Section 5 reports on our experiments, Section 6 contains some concluding remarks. Omitted proofs and other additional material can be found in the appendix of this paper [16].

2 Preliminaries

Throughout the paper, the reader is supposed to be familiar with the basic principles of ω-automata, games and temporal logics. For details we refer to [6,12]. We briefly summarize our notations for ω-automata, present the definition of good-for-games automata [13] and provide a condensed survey of the relevant principles of Markov decision processes (MDPs). Further details on MDPs and their use in the context of model checking can be found e.g. in [30,2].
Automata over infinite words. An ω-automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, Acc)$ is a tuple, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the (nondeterministic) transition function and $q_0 \in Q$ is the initial state. The last component Acc is the acceptance condition (see below). The size of $|\mathcal{A}|$ denotes the number of states in $\mathcal{A}$. $\mathcal{A}$ is said to be *complete*, if $\delta(q, \sigma) \neq \varnothing$ for all states $q \in Q$ and all symbols $\sigma \in \Sigma$. $\mathcal{A}$ is called *deterministic*, if $|\delta(q, \sigma)| \leq 1$ for all $q \in Q$ and $\sigma \in \Sigma$. A *run* in $\mathcal{A}$ for an infinite word $w = \sigma_0\, \sigma_1\, \sigma_2 \ldots \in \Sigma^\omega$ is a sequence $\rho = q_0\, q_1\, q_2 \ldots \in Q^\omega$ starting in the initial state q_0 such that $q_{i+1} \in \delta(q_i, a)$ for all $i \in \mathbb{N}$. We write $\inf(\rho)$ to denote the set of all states occurring infinitely often in ρ. A run ρ is called accepting, if it meets the acceptance condition Acc, denoted $\rho \models Acc$. We consider here the following three types of acceptance conditions and describe their constraints for infinite runs:

- *Büchi:* $Acc = F$ is a set of states, i.e., $F \subseteq Q$, with the meaning $\Box\Diamond F$
- *parity:* Acc is a function $col : Q \to \mathbb{N}$ assigning to each state q a parity color and requiring that the least parity color appearing infinitely often is even
- *Rabin:* Acc is a set consisting of pairs (E, F) with $E, F \subseteq Q$, imposing the constraint $\bigvee_{(E,F) \in Acc} (\Diamond\Box\neg E \wedge \Box\Diamond F)$

Büchi acceptance can be seen as a special case of parity acceptance which again can be seen as a special case of Rabin acceptance. We use the standard notations NBA (NRA, NPA) for nondeterministic Büchi (Rabin, parity) automata and DBA, DRA, DPA for their deterministic versions. The language of $\mathcal{A}$, denoted

$\mathcal{L}(\mathcal{A})$, consists of all infinite words $w \in \Sigma^\omega$ that have at least one accepting run in $\mathcal{A}$, i.e., $w \in \mathcal{L}(\mathcal{A})$ iff there exists a run ρ for w with $\rho \models Acc$.

It is well-known that the classes of languages recognizable by NBA, NRA, NPA, DRA or DPA are the same (the so-called ω-regular languages), while DBA are less powerful. For each LTL formula φ with atomic propositions in some finite set AP, the semantics of φ can be described as an ω-regular language $\mathcal{L}(\varphi)$ over the alphabet $\Sigma = 2^{AP}$ and there is an NBA $\mathcal{A}$ for φ (i.e., $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$) whose size is at most exponential in the formula length $|\varphi|$.

Good-for-games (GFG) automata. The formal definition of GFG automata [13] relies on a game-based view of ω-automata. Given an ω-automaton $\mathcal{A}$ as before, we consider $\mathcal{A}$ as the game arena of a turn-based two-player game, called *monitor game*: if the current state is q then player 1 chooses a symbol $\sigma \in \Sigma$ whereas the other player (player 0) has to answer by a successor state $q' \in \delta(q,\sigma)$, i.e., resolve the nondeterminism. In the next round q' becomes the current state. A *play* is an alternating sequence $\varsigma = q_0\,\sigma_0\,q_1\,\sigma_1\,q_2\,\sigma_2\,\ldots$ of states and (action) symbols in the alphabet Σ starting with the initial state q_0. Intuitively, the σ_i's are the symbols chosen by player 1 and the q_i's are the states chosen by player 0 in round i. Player 0 wins the play ς if ς is infinite and if $\varsigma|_\Sigma = \sigma_0\,\sigma_1\,\sigma_2\ldots \in \mathcal{L}(\mathcal{A})$ then $\varsigma|_Q = q_0\,q_1\,q_2\ldots$ is an accepting run. A strategy for player 0 is a function $\mathfrak{f} : (Q \times \Sigma)^+ \to Q$ with $\mathfrak{f}(\ldots q\,\sigma) \in \delta(q,\sigma)$. A play $\varsigma = q_0\,\sigma_0\,q_1\,\sigma_1\,q_2\ldots$ is said to be $\mathfrak{f}$-*conform* or a $\mathfrak{f}$-play if $q_i = \mathfrak{f}(\varsigma \downarrow i)$ for all $i \geq 1$ where $\varsigma \downarrow i = q_0\,\sigma_0\,\ldots\,\sigma_{i-2}\,\ldots q_{i-1}\sigma_i$ is the prefix of ρ that ends with the chosen symbol in round i. An automaton $\mathcal{A}$ is called *good-for-games* if there is a strategy $\mathfrak{f}$ such that player 0 wins each $\mathfrak{f}$-play. Such strategies will be called *GFG-strategies* for $\mathcal{A}$. Obviously, each deterministic automaton enjoys the GFG property. GFG automata with Rabin or parity condition cover the full class of ω-regular languages, while GFG automata with Büchi acceptance do not [4]. For illustrating examples of GFG automata see Appendix A.

Markov decision processes (MDP). MDPs are an operational model for systems that exhibit nondeterministic and probabilistic choices. For the purposes of this paper, we formalize an MDP by a tuple $\mathcal{M} = (S, Act, P, s_0, AP, \ell)$ where S is a finite set of states, $s_0 \in S$ is the initial state, Act a finite set of actions and $P : S \times Act \times S \to [0,1]$ is the transition probability function satisfying:

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

We write $Act(s)$ for the set of actions α that are enabled in s, i.e., $P(s,\alpha,s') > 0$ for some $s' \in S$, in which case $s' \mapsto P(s,\alpha,s')$ is a distribution formalizing the probabilistic effect of taking action α in state s. We refer to the triples (s,α,s') with $P(s,\alpha,s') > 0$ as a step. The choice between the enabled actions is viewed to be nondeterministic. For technical reasons, we require $Act(s) \neq \varnothing$ for all states s. The last two components AP and ℓ serve to formalize properties of paths in $\mathcal{M}$. Formally, AP is a finite set of atomic propositions and $\ell : S \to 2^{AP}$ assigns to each state s the set $\ell(s)$ of atomic propositions that hold in s. Paths in $\mathcal{M}$ are finite or infinite sequences $\pi = s_0\,\alpha_0\,s_1\,\alpha_2\,s_2\,\alpha_3\ldots$ starting in the initial state s_0 that are built by consecutive steps, i.e., $P(s_i,\alpha_i,s_{i+1}) > 0$ for all i. The trace of π is the word over the alphabet $\Sigma = 2^{AP}$ that arises by taking the projections to the state labels, i.e., $\mathit{trace}(\pi) = \ell(s_0)\,\ell(s_1)\,\ell(s_2)\ldots$. For an LTL formula φ over AP we write $\pi \models \varphi$ if $\mathit{trace}(\pi) \in \mathcal{L}(\varphi)$.

As the monitor game in nondeterministic automata, MDPs can be seen as stochastic games, also called a $1\frac{1}{2}$-player games. The first (full) player resolves the nondeterministic choice by selecting an enabled action α of the current state s. The second (half) player behaves probabilistically and selects a successor state s' with $P(s, \alpha, s') > 0$. Strategies for the full player are called *schedulers*. Since the behavior of $\mathcal{M}$ is purely probabilistic if some scheduler $\mathfrak{s}$ is fixed, one can reason about the probability of path events. If L is an ω-regular language then $\Pr^{\mathfrak{s}}_{\mathcal{M}}(L)$ denotes the probability under $\mathfrak{s}$ for the set of infinite paths π with $\mathit{trace}(\pi) \in L$. In notations like $\Pr^{\mathfrak{s}}_{\mathcal{M}}(\varphi)$ or $\Pr^{\mathfrak{s}}_{\mathcal{M}}(\mathcal{A})$ we identify LTL formulas φ and ω-automata $\mathcal{A}$ with their languages. For the mathematical details of the underlying sigma-algebra and probability measure, we refer to [30,2].

For a worst-case analysis of a system modeled by an MDP $\mathcal{M}$, one ranges over all initial states and all schedulers (i.e., all possible resolutions of the nondeterminism) and considers the maximal or minimal probabilities for some ω-regular language L. Depending on whether L represents a desired or undesired path property, the quantitative worst-case analysis amounts to computing $\Pr^{\min}_{\mathcal{M}}(\varphi) = \min_{\mathfrak{s}} \Pr^{\mathfrak{s}}_{\mathcal{M}}(L)$ or $\Pr^{\max}_{\mathcal{M}}(L) = \max_{\mathfrak{s}} \Pr^{\mathfrak{s}}_{\mathcal{M}}(L)$.

3 Automata-based Analysis of Markov Decision Processes

We address the task to compute the maximal or minimal probability in an MDP $\mathcal{M}$ for the path property imposed by a nondeterministic ω-automaton $\mathcal{A}$. The standard approach, see e.g. [2], assumes $\mathcal{A}$ to be deterministic and relies on a product construction where the transitions of $\mathcal{M}$ are simply annotated with the unique corresponding transition in $\mathcal{A}$. Thus, $\mathcal{M} \otimes \mathcal{A}$ can be seen as a refinement of $\mathcal{M}$ since $\mathcal{A}$ does not not affect $\mathcal{M}$'s behaviors, but attaches information on $\mathcal{A}$'s current state for the prefixes of the traces induced by the paths of $\mathcal{M}$.

We now modify the standard definition of the product for nondeterministic ω-automaton. The crucial difference is that the actions are now pairs $\langle \alpha, p \rangle$ consisting of an action in $\mathcal{M}$ and a state in $\mathcal{A}$, representing the nondeterministic alternatives in both the MDP $\mathcal{M}$ and the automaton $\mathcal{A}$. Formally, let $\mathcal{M} = (S, \mathit{Act}, P, s_0, AP, \ell)$ be an MDP and $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathit{Acc})$ a complete nondeterministic ω-automaton with $\Sigma = 2^{AP}$. The product MDP is

$$\mathcal{M} \otimes \mathcal{A} \ = \ (S \times Q, \mathit{Act} \times Q, P', \langle s_0, q_0 \rangle, AP, \ell')$$

where the transition probability function P' is given by $P'(\langle s, q \rangle, \langle \alpha, p \rangle, \langle s', q' \rangle) = P(s, \alpha, s')$ if $p = q' \in \delta(q, \ell(s))$. In all other cases $P'(\langle s, q \rangle, \langle \alpha, p \rangle, \langle s', q' \rangle) = 0$. The assumption that $\mathcal{A}$ is complete yields that for each $\alpha \in \mathit{Act}(s)$ there is some action $\langle \alpha, q' \rangle \in \mathit{Act}(\langle s, q \rangle)$ for all states s in $\mathcal{M}$ and q in $\mathcal{A}$. The labeling function is given by $\ell'(\langle s, q \rangle) = \{q\}$. Thus, the traces in $\mathcal{M} \otimes \mathcal{A}$ are words over the alphabet Q. Likewise, $\mathcal{A}$'s acceptance condition Acc can be seen as a language over Q, which permits to treat Acc as a property that the paths in $\mathcal{M} \otimes \mathcal{A}$ might or might not have. We prove in Appendix C:

Theorem 1. *For each MDP $\mathcal{M}$ and nondeterministic ω-automaton $\mathcal{A}$ as above:*

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

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

Theorem 1 (b) shows that with a slightly modified definition of the product, the techniques that are known for the quantitative analysis of MDPs against deterministic ω-automata specifications are also applicable for GFG automata. The computation of maximal probabilities for properties given by an ω-regular acceptance condition *Acc* (e.g., Büchi, Rabin or parity) can be carried out by a graph analysis that replaces *Acc* with a reachability condition and linear programming techniques for computing maximal reachability probabilities. See e.g. [2]. The time complexity is polynomial in the size of the $\mathcal{M}$ and $\mathcal{A}$. Thus, if the specification is given in terms of an LTL formula φ then the costs of our GFG-based approach are dominated by the generation of a GFG automaton for φ. Minimal probabilities can be handled by using $\Pr^{\min}_{\mathcal{M}}(\varphi) = 1 - \Pr^{\max}_{\mathcal{M}}(\neg\varphi)$.

[20,19] proves that a double exponential blow-up for translating LTL to deterministic ω-automata (of any type) is unavoidable. We adapted the proof in [19] for GFG automata (see App. D). Thus, the double exponential time complexity of the GFG-based approach is in accordance with the known 2EXPTIME-completeness for the analysis of MDPs against LTL specifications [7].

Theorem 2. *There exists a family of LTL formulas $(\varphi)_{n\in\mathbb{N}}$ such that $|\varphi_n| = \mathcal{O}(n)$, while every GFG automaton $\mathcal{A}_n$ for φ_n has at least $2^{2^{\Omega(n)}}$ states.*

4 From LTL to GFG Automata

We have previously shown [14,15] that it is possible in practice, using the tool LTL2DSTAR, to obtain deterministic ω-automata for a wide range of LTL formula φ via the translation to an NBA and Safra's determinization construction [31] refined by various heuristics. Here, we are interested in replacing Safra's determinization algorithm with the HP-algorithm [13] to generate a GFG automaton instead of a deterministic automaton. We first provide an outline of the HP-algorithm and then explain a few new heuristics.

The HP-algorithm transforms an NBA $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ with $|Q| = n$ states into a GFG automaton $\mathcal{A}$ with parity acceptance and at most $2^n \cdot n^{2n}$ states and $2n$ parity colors (or an NRA with n Rabin pairs), which improves on the upper bound given for Safra's determinization algorithm. We recall here the main concepts, for a formal description we refer to [13]. Like Safra's construction, the HP-algorithm relies on the simultaneous tracking of multiple subset constructions to determine acceptance or rejection in the NBA. However, while the states of Safra's DRA organize the subsets in trees, the HP-algorithm uses a simpler, linear arrangement of the subsets. The state space $P = (2^Q \times 2^Q)^n$ of the GFG automaton $\mathcal{A}$ consists of n pairs of subsets of NBA states Q, i.e., states of the form $p = \langle (A_1, B_1), \ldots, (A_n, B_n)\rangle$ where $B_i \subseteq A_i \subseteq Q$, plus some additional constraints on the state space. Each set B_i serves to mark those states in A_i

that were reached via some accepting state in F of the NBA. The successor state in $\mathcal{A}$ for symbol σ is obtained by applying the transition function δ to each of the subsets and adding states in F to the B_i subsets. In crucial difference to Safra's construction, the HP-algorithm however then introduces significant nondeterminism by allowing $\mathcal{A}$ to discard an arbitrary number of states in any of the subsets. For $p = \langle \ldots (A_i, B_i) \ldots \rangle$, the set A'_i in a σ-successor p' of $\mathcal{A}$ thus does not correspond to $A'_i = \delta(A_i, \sigma)$ but there is a nondeterministic choice between any A'_i satisfying $A'_i \subseteq \delta(A_i, \sigma)$, including the empty set. Whenever some A_i is empty, $\mathcal{A}$ can "reset" A_i by setting A_i to some subset of the first set A_1. Such resets are reflected in the acceptance condition of $\mathcal{A}$ as "bad" events for the pair i, as they signify that the previously tracked runs terminated. The "good" events in the acceptance condition occur whenever all states in an A_i are marked as having recently visited F, i.e., whenever $A_i = B_i \neq \varnothing$. In the next step, B'_i is then cleared and the tracking of visits to F starts anew. Infinitely many "good" events without "bad" events then correspond to the existence of an accepting run in the NBA $\mathcal{B}$. The HP-algorithm relies on the GFG-strategy to resolve the nondeterminism in the constructed automaton $\mathcal{A}$, i.e., which states in the subsets are kept, which are dropped and when to reset. There is a large amount of nondeterminism and a lot of combinatorial possibilities in the reachable state space of $\mathcal{A}$. This is confirmed by our experiments, e.g., applying the construction to the two-state NBA for $\Diamond\Box a$ already yields a GFG automaton with 16 states, where LTL2DSTAR generates a two-state DRA. As stated in [13], the HP-algorithm is thus not well-suited for an explicit representation for $\mathcal{A}$, but is intended for a symbolic implementation. In this context, [13] briefly discusses the possibility of variants of the transition function in the GFG automaton that either apply more or less strict constraints on the relationship enforced between the (A_i, B_i) pairs in each state. In particular, [13] posits that introducing even further nondeterminism (and increasing the number of possible states) by loosening a disjunctness requirement on the A_i may lead to a smaller symbolic representation. In our experiments, we will refer to this as the *loose variant*.

Iterative approach. In the context of games, [13] proposes an iterative approach to the HP-algorithm by successively constructing the automata $\mathcal{A}^m$ obtained by using only the first m of the n pairs, i.e., by setting $A_i = B_i = \varnothing$ for all $m < i \leq n$. In the acceptance condition this reduces the number of required parity colors to $2m$ and Rabin pairs to m as well. For these automata, $\mathcal{L}(\mathcal{A}^m) = \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$, but there is no guarantee that $\mathcal{A}^m$ for $m < n$ is good-for-games by construction. We start with $m = 1$ and increase m until early success or reaching $m = n$. Our experimental results indeed show that early termination appears rather often.

We now explain how the iterative approach of [13] can be integrated in the GFG-based quantitative analysis of MDPs against LTL specifications. Suppose, e.g., that the task is to show that $\Pr^{\max}_{\mathcal{M}}(\varphi) \geq \theta$ for some LTL formula φ and threshold $\theta \in]0,1]$. Let $\mathcal{B}$ be an n-state NBA with $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\varphi)$ and $\mathcal{A}^m$ the automaton obtained using only the first $m \leq n$ pairs in the HP-algorithm applied to $\mathcal{B}$. Let Acc^m denote the acceptance condition of $\mathcal{A}^m$. By Theorem 1 (a):

$$\text{If } \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}^m}\big(\, Acc^m \,\big) \geq \theta \text{ for some } m \leq n \text{ then } \Pr^{\max}_{\mathcal{M}}\big(\, \varphi \,\big) \geq \theta.$$

Moreover, $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}^m}\big(\,Acc^m\,\big) \le \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}^{m+1}}\big(\,Acc^{m+1}\,\big)$ for $m < n$. These observations suggest an approach that resembles the classical *abstraction-refinement* schema: starting with $m = 1$, we carry out the quantitative analysis of $\mathcal{M}\otimes\mathcal{A}^m$ against Acc^m and successively increase m until $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}^m}\big(\,Acc^m\,\big) \ge \theta$ or $\mathcal{A}^m$ is GFG (which is the case at the latest when $m = n$). As an additional heuristic to increase the performance of the linear programming techniques that are applied for the quantitative analysis of $\mathcal{M}\otimes\mathcal{A}^m$ against Acc^m, one can reuse the results computed for $\mathcal{M}\otimes\mathcal{A}^{m-1}$ and Acc^{m-1} as initial values.

It remains to explain how to check whether $\mathcal{A}^m$ has the GFG property. For details we refer to Appendix E. In this aspect, our prototype implementation departs from [13] and checks whether $\mathcal{A}^m$ is GFG by solving a Rabin game (itself an NP-complete problem) constructed from $\mathcal{A}^m$ and a DRA for $\neg\varphi$ constructed with LTL2DSTAR while [13] proposes an algorithm based on checking fair simulation. To study the impact of the iterative approach in terms of the number of required iterations and the size of the resulting GFG automata, the choice of the GFG test is irrelevant.

Union operator for disjunctive formulas. For generating a deterministic automaton from an LTL formula, we have shown in [14] that optionally handling disjunctive LTL formulas of the form $\varphi = \varphi_1 \vee \varphi_2$ by constructing DRA $\mathcal{A}_1$ and $\mathcal{A}_2$ for the subformulas φ_1 and φ_2 and then obtaining the DRA $\mathcal{A}_1 \cup \mathcal{A}_2$ for the language $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$ via a product construction can be very beneficial in practice. The definition of $\mathcal{A}_1 \cup \mathcal{A}_2$ used in [14] can easily be extended to NRA. The GFG property is preserved by the union construction. See Appendix F.

5 Implementation and Experiments

We have implemented the HP-algorithm in a tool we refer to as LTL2GFG. Based on LTL2GFG, we have additionally implemented the GFG-based quantitative analysis of MDPs in PRISM. After a brief overview of LTL2GFG, we report on our experiments and comparison with the determinization approach of LTL2DSTAR.

LTL2GFG. Given an LTL formula φ, our implementation LTL2GFG constructs a symbolic, BDD-based representation of a GFG-NPA for φ. It first converts φ into an (explicitly represented) NBA $\mathcal{B}$. In our experiments, we use LTL2BA v1.1 [11] for this task. To facilitate an efficient symbolic representation of the various subsets used in the HP-algorithm, $\mathcal{B}$ is then converted to a symbolic representation, using a unary encoding of the $|Q| = n$ states of $\mathcal{B}$, i.e., using one boolean variable q_i per state. The state space of the GFG-automaton $\mathcal{A}$, i.e., the n pairs (A_i, B_i) is likewise encoded by n^2 boolean variables $a_{i,j}$ and $b_{i,j}$, i.e., $a_{i,j}$ is true iff NBA state $q_j \in A_i$ and $b_{i,j}$ is true iff $q_j \in B_i$ for $1 \le i, j \le n$. To allow the encoding of the transition relations of $\mathcal{A}$ and $\mathcal{B}$, each state variable has a primed copy, i.e., q'_i, $a'_{i,j}$ and $b'_{i,j}$ and each of the k atomic proposition in φ is represented by a boolean variable l_i. For a BDD-based symbolic representation, the order of the variables is crucial. The state variables and their copies are always kept adjacent. The standard variable ordering used by LTL2GFG is then an interleaving of the $a_{i,j}$ and $b_{i,j}$ variables with the q_j variables, i.e.,

Table 1. Statistics for the automata $\mathcal{A}_\varphi$ constructed for the 94 benchmark formulas. Number of $\mathcal{A}_\varphi$ constructed within a given timeframe and a given range of BDD sizes.

		$\mathcal{A}_\varphi$ with constr. time				$\mathcal{A}_\varphi$ with BDD size					
	aborted	$<1s$	$<10s$	$<1m$	$<30m$	<10	$<10^2$	$<10^3$	$<10^4$	$<10^5$	$\geq 10^5$
LTL2DSTAR std.	0	90	91	92	94	4	65	87	90	91	3
no opt.	0	90	90	92	94	3	48	78	89	90	4
LTL2GFG std.	39	40	47	48	55	3	6	19	26	36	19
std., dynamic	45	34	36	48	49	5	8	19	36	39	10
loose, dynamic	34	43	49	56	60	5	14	31	47	56	4
lo., union, dyn.	29	52	59	61	65	4	13	35	54	60	5
lo., iterative	20	74	74	74	74	3	19	39	60	74	0
lo., it., un., dyn.	18	70	72	74	76	4	32	63	70	76	0

$l_1 < \ldots < l_k < q_1 < \ldots < q_j < a_{1,j} < b_{1,j} < a_{2,j} < b_{2,j} < \ldots < q_{j+1} < \ldots$. LTL2GFG uses the JINC C++ BDD library for the symbolic representation.

Experimental results for the HP-algorithm. We report here on a number of experiments with LTL2GFG using the benchmark formulas used in the evaluation of LTL2DSTAR in [14,15], i.e., 39 LTL formulas from the literature [10,33] and 55 pattern formulas [9] that represent common specification patterns. All our experiments were carried out on a computer with 2 Intel E5-2680 8-core CPUs at 2.70 GHz with 384GB of RAM running Linux and with a memory limit of 10 GB and a time-out of 30 minutes for each formula.

For the automata $\mathcal{A}_\varphi$, we report on the number of BDD nodes in the encoding of the transition function, as this the most crucial aspect. To allow a fair comparison with the explicit determinization in LTL2DSTAR, we consider symbolic encodings of the DRA $\mathcal{A}_\varphi$ obtained from LTL2DSTAR 0.5.1. This encoding uses $\lceil \log_2 n \rceil$ boolean variables to straightforwardly encode the n state indices in $\mathcal{A}_\varphi$, which is the same encoding employed in PRISM for its DRA-based approach to LTL model checking.

Table 1 presents statistics for the construction of DRA with LTL2DSTAR and GFG-NPA with LTL2GFG for the benchmark formulas. The LTL2DSTAR results are given once with standard settings and for a variant where all optimizations are disabled, i.e., with purely Safra's construction. For LTL2GFG, we start with the pure HP-algorithm and consider variants with the "loose" transition definition, the union construction, and with dynamic reordering of the variable order. We also give statistics for the iterative approach, where LTL2GFG constructs the partial automata $\mathcal{A}^m$ until it can be shown (via solving a Rabin game [29]) that the automaton is GFG.

LTL2DSTAR constructed most of the automata in a few seconds, the most difficult was constructed in $95s$ and had 1.2 million BDD nodes. Apart from the most difficult automata, the BDD sizes range in the hundreds and thousands. For all the LTL2GFG variants, a significant fraction of automata could not be constructed in the time and memory limits, around 40% for the standard HP-algorithm, and dropping to around 20% for the best variant. The loose variant by itself had a mixed effect, but in conjunction with dynamic reordering was

Table 2. Results of the iterative approach in LTL2GFG, for the loose variant. M is the minimal value $m \leq n$ for which the partial NPA $\mathcal{A}^m$ could be shown to be GFG.

	with n NBA states											
	2	3	4	5	6	7	8	9	10	11	12	>12
number of φ	13	17	13	9	8	3	3	1	4	2	4	11
number of φ, $M < n$	11	17	13	8	8	2	2	1	0	0	1	3
number of φ, $M = 1$	11	8	5	4	2	1	1	0	0	0	1	3
number of φ, $M = 2$	2	9	8	4	6	1	1	1	0	0	0	0
number of φ, GFG check aborted	0	0	0	1	0	1	1	0	4	2	3	8

generally beneficial. The union construction was very beneficial for the disjunctive formulas. For example, the automata for $\Box\Diamond a \to \Box\Diamond b$ could not be constructed in the time limits with the standard HP-algorithm but could be handled using the union construction. The iterative approach was successful as well in obtaining smaller automata, which is explained by the fact that for a large number of formulas it could be shown that the partial automata $\mathcal{A}^1$ or $\mathcal{A}^2$ were already GFG, as detailed in Table 2. For the iterative approach we were mostly focused on experimental data for the minimal value m for which $\mathcal{A}^m$ becomes GFG and the effect on the BDD size. Different algorithms or implementations for the GFG check than the one used in LTL2GFG lead to the same final GFG automata, but could improve the performance. At the end, despite the various approaches implemented in LTL2GFG, there were only 6 formulas with relatively small automata where the BDD size of the smallest GFG automaton was smaller than that of the DRA obtained from LTL2DSTAR (172 nodes instead of 229 nodes, 219 instead of 347, and the other 4 automata differing by 1 or 2 at a size of less than 20 nodes). We do not report here in detail on the number of reachable states in the automata, as none of the GFG automata had a smaller number of states than the DRA generated by LTL2DSTAR. In particular, the automata obtained without the iterative approach often had millions and more states.

Implementation in PRISM. We have extended the MTBDD-based, symbolic engines of PRISM 4.1 with an implementation of our algorithm for computing $\Pr^{\max}_{\mathcal{M}}(\varphi)$ using GFG automata for φ (and $\Pr^{\min}_{\mathcal{M}}(\varphi)$ using a GFG automaton for $\neg\varphi$). We import the BDD of $\mathcal{A}$ generated with LTL2GFG into PRISM and perform the product with $\mathcal{M}$ and analysis in $\mathcal{M} \otimes \mathcal{A}$ symbolically. In its standard approach, PRISM constructs an explicit DRA with an integrated version of LTL2DSTAR, which is then symbolically encoded as described before. The analysis is then carried out symbolically as well.

Experiments in PRISM. As a benchmark, we used a PRISM model [23] for parts of the WLAN carrier-sense protocol of IEEE 802.11. As was to be expected given our results on the automata construction, the GFG-based analysis did not improve on the standard approach. Even using the optimal variant of LTL2GFG for each formula, ignoring the automata construction times, and for cases where the product $\mathcal{M} \otimes \mathcal{A}$ had a comparable BDD size for the GFG- and DRA-based approach, the model checking using the GFG automata took significantly longer. For further details, we refer to Appendix G.

6 Conclusion

We have shown that GFG automata can replace deterministic automata for the quantitative analysis of MDPs against ω-regular specifications without increasing the asymptotic worst-case time complexity. To evaluate the GFG-based approach from the practical side, we implemented the HP-algorithm, integrated several heuristics, and performed exhaustive experiments for the LTL-to-GFG construction and for probabilistic model checking. Our experimental results are a bit disappointing, as the generated GFG automata were often larger than DRA generated by the implementation of Safra's algorithms in LTL2DSTAR, both in the number of states and in the symbolic BDD-based representations. Thus, our empirical results are in contrast to the expectation that the HP-algorithm yields GFG automata that are better suited for symbolic approaches rather than DRA generated by Safra's algorithm. Also in the context of probabilistic model checking, the GFG-based approach turned out to be more time- and memory-consuming than the traditional approach with deterministic automata. However, it is still too early to discard the concept of GFG automata for practical purposes. Our negative empirical results might be an artefact of the HP-algorithm, which is – to the best of our knowledge – the only known algorithm for the generation of GFG automata that are not deterministic. Future directions are the design of other algorithms for the construction of succinct GFG automata. Alternatively, one might seek for automata types that are still adequate for probabilistic model checking and other areas, but rely on weaker conditions than the GFG property.

References

1. Babiak, T., Blahoudek, F., Kretínský, M., Strejcek, J.: Effective translation of LTL to deterministic Rabin automata: beyond the (F,G)-Fragment. In: ATVA'13. LNCS, vol. 8172, pp. 24–39. Springer (2013)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
3. Benedikt, M., Lenhardt, R., Worrell, J.: Two variable vs. linear temporal logic in model checking and games. Logical Methods in Computer Science 9(2) (2013)
4. Boker, U., Kuperberg, D., Kupferman, O., Skrzypczak, M.: Nondeterminism in the presence of a diverse or unknown future. In: ICALP'13. LNCS, vol. 7966, pp. 89–100. Springer (2013)
5. Bustan, D., Rubin, S., Vardi, M.: Verifying ω-regular properties of Markov chains. In: CAV'04. LNCS, vol. 3144, pp. 189–201. Springer (2004)
6. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (2000)
7. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. Journal of ACM 42(4), 857–907 (1995)
8. Couvreur, J.M., Saheb, N., Sutre, G.: An optimal automata approach to LTL model checking of probabilistic systems. In: LPAR'03. LNCS, vol. 2850, pp. 361–375. Springer (2003)
9. Dwyer, M., Avrunin, G., Corbett, J.: Patterns in property specifications for finite-state verification. In: ICSE'99. pp. 411–420. ACM (1999)
10. Etessami, K., Holzmann, G.: Optimizing Büchi automata. In: CONCUR'00. LNCS, vol. 1877, pp. 153–167. Springer (2000)

11. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: CAV'01. LNCS, vol. 2102, pp. 53–65. Springer (2001)
12. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research, LNCS, vol. 2500. Springer (2002)
13. Henzinger, T., Piterman, N.: Solving games without determinization. In: CSL'06. LNCS, vol. 4207, pp. 395–410. Springer (2006)
14. Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theoretical Computer Science 363(2), 182–195 (2006)
15. Klein, J., Baier, C.: On-the-fly stuttering in the construction of deterministic ω-automata. In: CIAA'07. LNCS, vol. 4783, pp. 51–61. Springer (2007)
16. Klein, J., Müller, D., Baier, C., Klüppelholz, S.: Are good-for-games automata good for probabilistic model checking? (extended version). Tech. rep., Technische Universität Dresden (2013), `http://wwwtcs.inf.tu-dresden.de/ALGI/PUB/LATA14/`
17. Kretínský, J., Ledesma-Garza, R.: Rabinizer 2: Small deterministic automata for LTL$\setminus$GU. In: ATVA'13. LNCS, vol. 8172, pp. 446–450. Springer (2013)
18. Kupferman, O., Piterman, N., Vardi, M.: Safraless compositional synthesis. In: CAV'06. LNCS, vol. 4144, pp. 31–44. Springer (2006)
19. Kupferman, O., Rosenberg, A.: The blowup in translating LTL to deterministic automata. In: MoChArt'10. LNCS, vol. 6572, pp. 85–94. Springer (2011)
20. Kupferman, O., Vardi, M.: From linear time to branching time. ACM Transactions on Computational Logic 6(2), 273–294 (2005)
21. Kupferman, O., Vardi, M.: Safraless decision procedures. In: FOCS'05. pp. 531–542. IEEE Computer Society (2005)
22. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. International Journal on Software Tools for Technology Transfer (STTT) 6(2), 128–142 (2004)
23. Kwiatkowska, M., Norman, G., Sproston, J.: Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In: PAPM-PROBMIV'02. LNCS, vol. 2399, pp. 169–187. Springer (2002)
24. Latvala, T.: Efficient model checking of safety properties. In: SPIN'03. LNCS, vol. 2648, pp. 74–88. Springer (2003)
25. Löding, C.: Optimal bounds for transformations of omega-automata. In: FSTTCS'99. LNCS, vol. 1738, pp. 97–109. Springer (1999)
26. Michel, M.: Complementation is more difficult with automata on infinite words (1988), CNET, Paris
27. Morgenstern, A., Schneider, K.: From LTL to symbolically represented deterministic automata. In: VMCAI'08. LNCS, vol. 4905, pp. 279–293. Springer (2008)
28. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Logical Methods in Computer Science 3(3:5), 1–21 (2007)
29. Piterman, N., Pnueli, A.: Faster solutions of Rabin and Streett games. In: LICS'06. pp. 275–284. IEEE (2006)
30. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY (1994)
31. Safra, S.: On the complexity of ω-automata. In: FOCS. pp. 319–327. IEEE (1988)
32. Schewe, S.: Tighter bounds for the determinisation of Büchi automata. In: FOSSACS'09. LNCS, vol. 5504, pp. 167–181. Springer (2009)
33. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: CAV'00. LNCS, vol. 1855, pp. 248–263. Springer (2000)
34. Vardi, M., Wolper, P.: An automata-theoretic approach to automatic program verification. In: LICS'86. pp. 332–344. IEEE Computer Society (1986)

Appendix

A Examples for Good-for-games Automata

In this section we want to provide an intuition for the GFG property by presenting two examples.

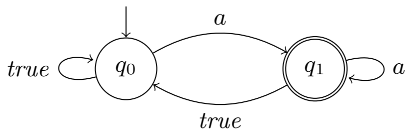


Fig. 1. Good-for-games NBA $\mathcal{A}$ for $\Box \Diamond a$

The first example, depicted in Figure 1, is an good-for-games Büchi automaton. It accepts the language $\mathcal{L}(\Box \Diamond a)$. This automaton is nondeterministic for the symbols that contain a, allowing the choice of either staying in the current state or switching to the other. A winning strategy for player 0 is the following strategy $\mathfrak{f}$: Whenever player 0 is able to choose moving to state q_1, he should do so. This means that if the current state of the play is state q_1 and player 1 choose a symbol where a does not occur, then player 0 chooses state q_0. In all other cases player 0 moves to q_1.

This strategy ensures that all words accepted by $\mathcal{L}(\mathcal{A})$ are indeed accepted, as the only way for player 1 to generate a word $w \in \mathcal{L}(\mathcal{A})$ is to choose infinitely many symbols with a. But then player 0 visits infinitely often the accepting state q_1, since every time a symbol with a is selected, player 0 moves to q_1 via his strategy. So for an accepted word the $\mathfrak{f}$-conform play contains infinitely many accepting states q_1 and thus the projection to the automata states delivers an accepting run.

The second NBA, shown in Figure 2, is not good-for-games. Let player 1 pick the symbol $\{a\}$ in the beginning. Now there exists a nondeterministic choice between q_1 and q_2. Assume that the strategy for player 0 moves to q_1. Then, player 1 can choose the symbol $\{c\}$, leading to the trap state q_t from which no accepting run is possible. However, every word starting with $\{a\}\,\{c\}$ is an accepted word and thus choosing q_1 in the first choice is not a valid move for a winning strategy. In the other case, if the strategy of player 0 moves to q_2 after the first symbol $\{a\}$, player 1 can choose the symbol $\{b\}$, trapping the run in the nonaccepting q_t again. So, for every accepted word $\{a\}\,\{b\}\ldots$ the conform play again does not yield an accepted run and thus choosing q_2 is not a valid move for a winning strategy either.

As a whole, we get that there does not exists a winning strategy, because in the initial state q_0 player 0 would need to know which of the two symbol player 1 will choose in the second step. So player 0 would need the ability to *look-ahead*, which is not allowed for GFG automata.

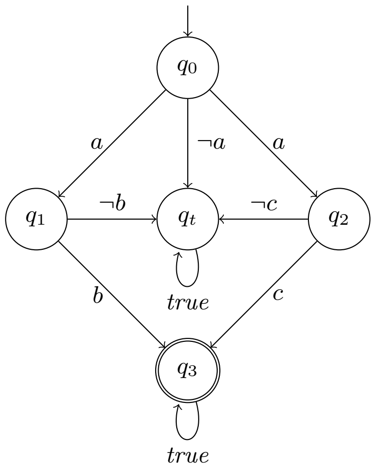
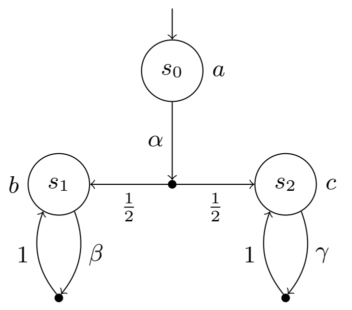


Fig. 2. NBA $\mathcal{B}$ for $a \wedge \bigcirc(b \vee c)$, not good-for-game

B Necessity of the GFG Property in Theorem 1(b)

To illustrate that the GFG-property is crucial in part (b) of Theorem 1, we provide an example for an MDP $\mathcal{M}$ (even Markov chain) and nondeterministic ω-automaton $\mathcal{A}$ such that $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(\mathit{Acc})$ is strictly smaller than $\Pr^{\max}_{\mathcal{M}}(\mathcal{A})$.



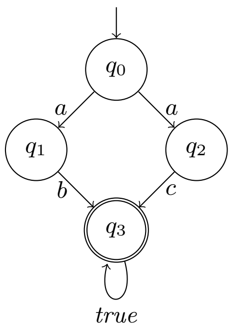


Fig. 3. Markov decision process $\mathcal{M}$ (left) and NBA $\mathcal{A}$ for $a \wedge \bigcirc(b \vee c)$ (right)

The left of Figure 3 shows the MDP $\mathcal{M}$ with three states s_0, s_1, s_2, actions α, β and γ and atomic propositions a, b and c. $\mathcal{M}$ behaves purely probabilistically. Hence, it can be seen as a Markov chain and the concept of schedulers is irrelevant for $\mathcal{M}$. As $\mathcal{M}$ has only two paths $\pi_1 = s_0\,\alpha\,s_1\,\beta\,s_1\,\beta\,s_1\,\beta\ldots$ and $\pi_2 = s_0\,\alpha\,s_2\,\gamma\,s_2\,\gamma\,s_2\,\gamma\ldots$, $\mathcal{M}$ has only two traces, namely $\{a\}\,\{b\}^\omega$ and $\{a\}\,\{c\}^\omega$.

The picture on the right shows a fragment of an NBA $\mathcal{A}$ over the alphabet $2^{\{a,b,c\}}$ and the accepting state q_3. Thus, *Acc* is given by $\Box\Diamond q_3$. To ensure that $\mathcal{A}$ is complete, $\mathcal{A}$ has an additional trap state q_t (not shown in Figure 4). For every state q and every symbol σ, if no transition with symbol σ starting in q is depicted, then $\delta(q,\sigma) = \{q_t\}$. Clearly, $\mathcal{L}(\mathcal{A})$ is the language for the LTL formula

$$\varphi \;=\; a \wedge \bigcirc(b \vee c)$$

Since both paths π_1 and π_2 of $\mathcal{M}$ satisfy φ, the probability for φ in $\mathcal{M}$ is 1, which yields $\Pr^{\max}_{\mathcal{M}}(\mathcal{A}) = 1$.

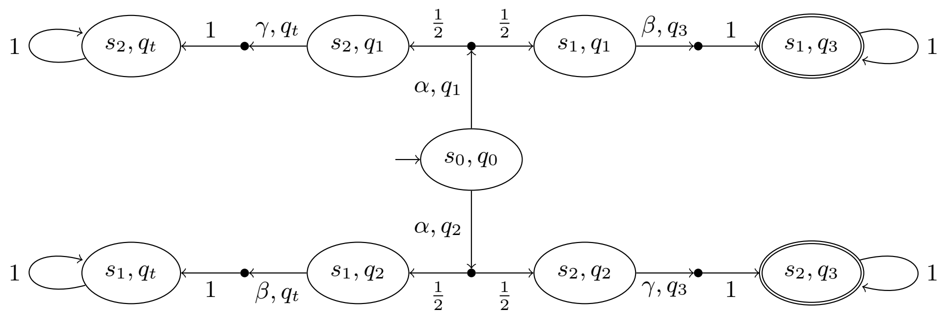


Fig. 4. Product MDP $\mathcal{M} \otimes \mathcal{A}$

Fig. 4 shows the product-MDP $\mathcal{M} \otimes \mathcal{A}$. Any scheduler $\mathfrak{s}$ for $\mathcal{M} \otimes \mathcal{A}$ just has two options in the initial state $\langle s_0, q_0\rangle$ "select action $\langle \alpha, q_1\rangle$ or action $\langle \alpha, q_2\rangle$", while there is just one enabled action for the other states. In particular, $\mathcal{M} \otimes \mathcal{A}$ has just two schedulers.

By symmetry, it suffices to consider the scheduler $\mathfrak{s}$ choosing action $\langle \alpha, q_1\rangle$ for the initial state. The $\mathfrak{s}$-paths resolve the probabilistic choice between $\langle s_1, q_1\rangle$ and $\langle s_2, q_1\rangle$. No accepting state of the form $\langle s, q_3\rangle$ is reachable from state $\langle s_2, q_1\rangle$, while the accepting state $\langle s_1, q_3\rangle$ will be reached in the next step from $\langle s_1, q_1\rangle$. Hence:

$$\Pr^{\mathfrak{s}}_{\mathcal{M}\otimes\mathcal{A}}(\,Acc\,) \;=\; \frac{1}{2}$$

The same argument applies to the scheduler for $\mathcal{M} \otimes \mathcal{A}$ that chooses action $\langle \gamma, q_2\rangle$ in the first step. Thus, we get:

$$\frac{1}{2} \;=\; \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}(Acc) \;<\; \Pr^{\max}_{\mathcal{M}}(\mathcal{A}) \;=\; 1$$

C Proof of Theorem 1 and Minimal Probabilities

The goal is to show that for each MDP $\mathcal{M}$ and complete nondeterministic ω-automaton $\mathcal{A}$:

(a) $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}\big(\mathit{Acc}\big) \ \leq \ \Pr^{\max}_{\mathcal{M}}\big(\mathcal{A}\big)$

(b) If $\mathcal{A}$ is good-for-games then: $\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}\big(\mathit{Acc}\big) \ = \ \Pr^{\max}_{\mathcal{M}}\big(\mathcal{A}\big)$

We first observe that by the definition of the transition probability function P' we have:

- If $\pi' = \langle s_0, q_0\rangle\,\gamma_0\,\langle s_1, q_1\rangle\,\gamma_1\,\langle s_2, q_2\rangle\,\gamma_2\ldots$ is a path in $\mathcal{M}\otimes\mathcal{A}$ where $\gamma_i = \langle \alpha_i, p_i\rangle$ then $p_i = q_{i+1}$ and $\pi'|_{\mathcal{M}} = s_0\,\alpha_0\,s_1\,\alpha_1\,s_2\,\alpha_2\ldots$ is a path in $\mathcal{M}$ and $\pi'|_{\mathcal{A}} = q_0\,q_1\,q_2\ldots$ is a run in $\mathcal{A}$ for the word

$$\mathit{trace}\big(\pi'|_{\mathcal{M}}\big) \quad = \quad \ell(s_0)\,\ell(s_1)\,\ell(s_2)\ldots \in \big(2^{AP}\big)^{\omega}$$

  In this case, we have:

$$\pi' \models \mathit{Acc} \quad \text{iff} \quad \text{the run } \pi'|_{\mathcal{A}} \text{ is accepting}$$

- Vice versa, if $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,s_2\,\alpha_2\ldots$ is a path in $\mathcal{M}$ and $\rho = q_0\,q_1\,q_2\ldots$ a run in $\mathcal{A}$ for its trace then

$$\pi_\rho \quad = \quad \langle s_0, q_0\rangle\,\gamma_0\,\langle s_1, q_1\rangle\,\gamma_1\,\langle s_2, q_2\rangle\,\gamma_2\ldots$$

  is a path in $\mathcal{M}\otimes\mathcal{A}$ where $\gamma_i = \langle \alpha_i, q_{i+1}\rangle$. In this case, we have: ρ is accepting iff $\pi_\rho \models \mathit{Acc}$.

Proof of statement (a). We pick a scheduler $\mathfrak{s}'$ for $\mathcal{M}\otimes\mathcal{A}$ that maximizes the probability for $\mathcal{A}$'s acceptance condition. The goal is to derive a scheduler $\mathfrak{s}$ for $\mathcal{M}$ under which the probability for generating traces in $\mathcal{L}(\mathcal{A})$ is at least

$$\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}\big(\mathit{Acc}\big) \quad = \quad \Pr^{\mathfrak{s}'}_{\mathcal{M}\otimes\mathcal{A}}\big(\mathit{Acc}\big).$$

The task is now to define $\mathfrak{s}(\pi) \in \mathit{Act}(\,\mathit{last}(\pi)\,)$ for finite paths π in $\mathcal{M}$ where $\mathit{last}(\pi)$ denotes the last state of π. Let $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\alpha_{n-1}\,s_n$ be a finite path in $\mathcal{M}$. We introduce inductively states $q_1,\ldots,q_n,q_{n+1}$ in $\mathcal{A}$ as follows. Let

$$\gamma_0 \ \stackrel{\text{def}}{=} \ \langle \alpha_0, q_1\rangle \ = \ \mathfrak{s}'(\langle s_0, q_0\rangle)$$

and for $0 \leq i \leq n$:

$$\gamma_i \ \stackrel{\text{def}}{=} \ \langle \alpha_i, q_{i+1}\rangle \ = \ \mathfrak{s}'\big(\,\langle s_0, q_0\rangle\,\gamma_0\,\langle s_1, q_1\rangle\,\gamma_1\,\ldots\,\gamma_{i-1}\,\langle s_i, q_i\rangle\,\big)$$

Clearly, in the above inductive definition we have $q_{i+1} \in \delta(q_i, \ell(s_i))$ and $\alpha_i \in \mathit{Act}(s_i)$. We then define:

$$\mathfrak{s}(s_0\,\alpha_0\,s_1\,\alpha_1\ldots\alpha_{n-1}\,s_n) \ \stackrel{\text{def}}{=} \ \alpha_n$$

Suppose now that $\pi' = \langle s_0, q_0\rangle\,\gamma_0\,\langle s_1, q_1\rangle\,\gamma_1\,\langle s_2, q_2\rangle\,\gamma_2\ldots$ is an infinite $\mathfrak{s}'$-path with $\pi' \models \mathit{Acc}$. Then, the run $\pi'|_{\mathcal{A}} = q_0\,q_1\,q_2\ldots$ for the word $\mathit{trace}\big(\pi'|_{\mathcal{M}}\big) = \ell(s_0)\,\ell(s_1)\,\ell(s_2)\ldots$ is accepting. Thus, the set of all $\mathfrak{s}$-paths π with $\mathit{trace}(\pi) \in$

$\mathcal{L}(\mathcal{A})$ contains the set of the paths $\pi'|_{\mathcal{M}}$ where π' is a $\mathfrak{s}'$-path in $\mathcal{M} \otimes \mathcal{A}$ with $\pi' \models Acc$. We obtain the desired result:

$$\Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}\big(Acc\big) \;=\; \Pr^{\mathfrak{s}'}_{\mathcal{M}\otimes\mathcal{A}}\big(Acc\big) \;\leq\; \Pr^{\mathfrak{s}}_{\mathcal{M}}\big(\mathcal{A}\big) \;\leq\; \Pr^{\max}_{\mathcal{M}}\big(\mathcal{A}\big)$$

Proof of statement (b). We suppose now that $\mathcal{A}$ is good-for-games. By (a), it suffices to show that

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

Let $\mathfrak{f}$ denote a GFG-strategy for the monitor game for $\mathcal{A}$ and let $\mathfrak{s}$ be a scheduler in $\mathcal{M}$ that maximizes the probability to generate traces in $\mathcal{L}(\mathcal{A})$. The goal is to compose $\mathfrak{s}$ and $\mathfrak{f}$ to obtain a scheduler $\mathfrak{s}'$ for $\mathcal{M} \otimes \mathcal{A}$ such that the probability under $\mathfrak{s}'$ for the paths π' with $\pi' \models Acc$ is at least $\Pr^{\mathfrak{s}}_{\mathcal{M}}\big(Acc\big)$.

The definition of $\mathfrak{s}'(\pi')$ for the finite paths π' in $\mathcal{M} \otimes \mathcal{A}$ is by induction on the length of π'. For the initial state we define:

$$\mathfrak{s}'(\langle s_0, q_0\rangle) \;\stackrel{\text{def}}{=}\; \langle\, \mathfrak{s}(s_0),\, \mathfrak{f}(q_0\, \ell(s_0))\,\rangle$$

For a finite path $\pi' = \langle s_0, q_0\rangle\, \gamma_0\, \langle s_1, q_1\rangle\, \gamma_1\, \ldots \gamma_{n-1}\, \langle s_n, q_n\rangle$ in $\mathcal{M} \otimes \mathcal{A}$ of length $n \geq 1$ where $\gamma_i \;=\; \langle \alpha_i, q_{i+1}\rangle$, the definition of $\mathfrak{s}(\pi')$ is as follows:

$$\mathfrak{s}'(\pi') \;\stackrel{\text{def}}{=}\; \big\langle\, \mathfrak{s}(\pi'|_{\mathcal{M}}),\, \mathfrak{f}\big(q_0\, \ell(s_0)\, q_1\, \ell(s_1)\, \ldots\, q_{n-1}\, \ell(s_{n-1})\big)\,\big\rangle$$

Suppose now that $\pi \;=\; s_0\, \alpha_0\, s_1\, \alpha_1\, s_2\, \alpha_2 \ldots$ is an infinite $\mathfrak{s}$-path in $\mathcal{M}$ with $trace(\pi) \in \mathcal{L}(\mathcal{A})$. We now consider the accepting run $\rho = q_0\, q_1\, q_2\, \ldots$ for $trace(\pi)$ that is obtained using the GFG-strategy $\mathfrak{f}$ in the monitor game for $\mathcal{A}$. That is:

$$q_{i+1} \;=\; \mathfrak{f}\big(\, q_0\, \ell(s_0)\, q_1\, \ell(s_1)\, \ldots q_i\, \ell(s_i)\,\big)$$

Then, $\pi_\rho \;=\; \langle s_0, q_0\rangle\, \gamma_0\, \langle s_1, q_1\rangle\, \gamma_1\, \langle s_2, q_2\rangle\, \gamma_2\, \ldots$ is an infinite $\mathfrak{s}'$-path with $\pi_\rho \models Acc$ where $\gamma_i \;=\; \langle \alpha_i, q_{i+1}\rangle$. Thus, the set of all infinite $\mathfrak{s}'$-paths π' with $\pi' \models Acc$ subsumes all paths π_ρ resulting from combining a $\mathfrak{s}$-path π in $\mathcal{M}$ where $trace(\pi) \in \mathcal{L}(\mathcal{A})$ with its (unique) accepting $\mathfrak{f}$-run ρ. This yields:

$$\Pr^{\max}_{\mathcal{M}}\big(\mathcal{A}\big) = \Pr^{\mathfrak{s}}_{\mathcal{M}}\big(\mathcal{A}\big) \leq \Pr^{\mathfrak{s}'}_{\mathcal{M}\otimes\mathcal{A}}\big(Acc\big) \leq \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}\big(Acc\big)$$

This completes the proof of statement (b) in Theorem 1.

Minimal probabilities. If $\mathcal{M}$ is an MDP as before and $\mathfrak{s}$ a scheduler for $\mathcal{M}$ then for each ω-regular language L over the alphabet 2^{AP}:

$$\Pr^{\mathfrak{s}}_{\mathcal{M}}(L) \;=\; 1 - \Pr^{\mathfrak{s}}_{\mathcal{M}}(\overline{L})$$

where $\overline{L}$ denotes the complement of L, i.e., $\overline{L} = \big(2^{AP}\big)^{\omega} \setminus L$. Hence, we get:

$$\Pr^{\min}_{\mathcal{M}}(L) \;=\; 1 - \Pr^{\max}_{\mathcal{M}}(\overline{L})$$

As a consequence of Theorem 1, we obtain:

Corollary 3. *For each MDP $\mathcal{M}$, LTL formula φ and complete GFG-automaton $\mathcal{A}$ for $\neg\varphi$:*

$$\Pr^{\min}_{\mathcal{M}}\big(\,\varphi\,\big) \;=\; 1 - \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}\big(Acc\big)$$

Moreover, $\Pr^{\min}_{\mathcal{M}}\big(\,\varphi\,\big) \leq 1 - \Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}}\big(Acc\big)$ for each nondeterministic ω-automaton $\mathcal{A}$ for $\neg\varphi$.

D Proof of Theorem 2

In [20,19], a lower bound of $2^{2^{\Omega(|\varphi|)}}$ is shown for the translation from LTL formulas φ to deterministic Büchi automata (and thus for the other types of deterministic ω-automata as well). Let $\Sigma = \{\sigma_1, \sigma_2\}$ be an alphabet with two elements. [20] defines a family of languages L_n over the alphabet $\Sigma \cup \{\#, \$\}$ of the following form:

$$L_n \;=\; \bigcup_{w \in \Sigma^n} L(w)$$

where $L(w)$ is the following ω-regular language:

$$L(w) \;=\; \left\{\; x \,\#\, w \,\#\, y \,\$\, w \,\#^\omega \;:\; x, y \in (\Sigma \cup \{\#\})^* \;\right\}$$

The languages L_n are safety languages and can thus be recognized by a DBA $\mathcal{A}_n$, which needs at least $2^{2^{\Omega(n)}}$ states. Intuitively, this is because $\mathcal{A}_n$ needs to check that the word $w \in \{a,b\}^n$ after the \$ symbol has appeared before. The proof from [20], that every DBA accepting the language L_n needs at least $2^{2^{\Omega(n)}}$ states can be extended to a GFG automata in a straightforward manner.

Lemma 4. *Every good-for-games Büchi automaton $\mathcal{A}_n$ recognising L_n has at least 2^{2^n} states.*

Proof. Proof by contradiction. We refer to words in Σ^n as n-blocks. Further we assume a total order on Σ^n, thereby we can define w_i as the i-th word in Σ^n. Additionally, we define for every set

$$I \;=\; \{i_1, i_2, \ldots, i_k\} \;\subseteq\; \{0, 1, \ldots, 2^n - 1\}$$

the finite word $w_I = \#w_{i_1}\#w_{i_2}\#\ldots\#w_{i_k}\#\$$. As $\mathcal{A}_n$ is good-for-games, there exists a GFG-strategy $\mathfrak{f}$. Let q_I be the state reached in the $\mathfrak{f}$-play after consuming the finite word w_I. There exists 2^{2^n} subsets of $\{0, 1, \ldots, 2^n - 1\}$, but by assumption $\mathcal{A}_n$ has less than 2^{2^n} states. Therefore there must be two distinct sets $I \neq J$ with $q_I = q_J$. W.l.o.g. let $i \in I \setminus J$. The word $w_I\, w_i\, \#^\omega$ belongs to L_n, and hence is accepted by $\mathcal{A}_n$ and the $\mathfrak{f}$-play for $w_I\, w_i\, \#^\omega$ is accepting as well. On the other hand, $w_J\, w_i\, \#^\omega$ is not in L_n. But as $q_I = q_J$ there is an accepting run for the suffix $w_i\, \#^\omega$ and we get $w_J\, w_i\, \#^\omega \in L_n$. Contradiction.

The same argument holds for the other acceptance conditions, e.g., Rabin or parity acceptance, as well. As shown in [20], there is an LTL formula φ_n of size $\mathcal{O}(n^2)$ for L_n, which is improved in [19] to an LTL formula φ_n of size $\mathcal{O}(n)$, yielding the double exponential lower bound.

E Game-based Characterization of the GFG Property

We provide here a game-based characterization of the GFG property, which can serve as basis to check whether a given ω-automaton $\mathcal{A}$ is good-for-games. This algorithm for checking the GFG property will be used in our implementation of the iterative approach of the HP-construction.

Given a NRA $\mathcal{A} = (Q, \Sigma, \delta_{\mathcal{A}}, q_0, Acc_{\mathcal{A}})$ then – by definition – $\mathcal{A}$ is good-for-games, if there is a strategy that generates an accepting run for exactly the words w with $w \in \mathcal{L}(\mathcal{A})$. Since the class of ω-regular languages is closed under complementation there exists a DRA $\mathcal{B} = (P, \Sigma, \delta_{\mathcal{B}}, p_0, Acc_{\mathcal{B}})$ with $\mathcal{L}(\mathcal{B}) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$. W.l.o.g., we assume transition relations of $\mathcal{A}$ and $\mathcal{B}$ to be total.

We now construct a turn-based two-player game $\mathcal{G}_{\mathcal{A},\mathcal{B}}$ with full observation for both players as follows. The set of game vertices is $V = V_1 \cup V_0$ where the set of vertices where player 1 moves is $V_1 = Q \times P$ and where player 0 moves in the vertices in $V_0 = Q \times P \times \Sigma$. We set the initial vertex to $\langle q_0, p_0 \rangle$. The moves in $\mathcal{G}_{\mathcal{A},\mathcal{B}}$ are defined by the following two SOS-rules:

$$\frac{q \in Q, p \in P, \sigma \in \Sigma}{\langle q, p \rangle \longrightarrow \langle q, p, \sigma \rangle}$$

and

$$\frac{q' \in \delta_{\mathcal{A}}(q, \sigma),\ p' = \delta_{\mathcal{B}}(p, \sigma)}{\langle q, p, \sigma \rangle \longrightarrow (q', p')}$$

Player 1 chooses a symbol $\sigma \in \Sigma$ and player 0 resolves the nondeterminism. As $\mathcal{B}$ is deterministic, the game-structure $\mathcal{G}_{\mathcal{A},\mathcal{B}}$ can be viewed as a refinement of the monitor game associated with $\mathcal{A}$. The states in $\mathcal{A}$ are simply augmented with the information on $\mathcal{B}$'s current state and the chosen symbol.

The objective in $\mathcal{G}_{\mathcal{A},\mathcal{B}}$ is defined such that player 0 wins a play ς, if for every word w:

– if $w \in \mathcal{L}(\mathcal{A})$ then $\varsigma|_{\mathcal{A}}$ is an accepting run in $\mathcal{A}$
– if $w \notin \mathcal{L}(\mathcal{A})$ then $\varsigma|_{\mathcal{B}}$ is an accepting run in $\mathcal{B}$.

where $\varsigma|_{\mathcal{A}}$ denotes the projection of ς to the states $\mathcal{A}$. More precisely, we erase all vertices $\langle q, p, \sigma \rangle$ from ς and replace the vertices $\langle q, p \rangle$ with q. Likewise $\varsigma|_{\mathcal{B}}$ arises from ς by taking the projection to the $\mathcal{B}$-components of the vertices $\langle q, p \rangle$ in ς. Formally, the objective for player 0 in the game $\mathcal{G}_{\mathcal{A},\mathcal{B}}$ is the Rabin condition resulting from the union of the acceptance conditions of $\mathcal{A}$ and $\mathcal{B}$ lifted to the product. That is, if

$$\begin{aligned} Acc_{\mathcal{A}} &= \bigvee_{1 \le i \le n} \Diamond \Box \neg E_i^{\mathcal{A}} \wedge \Box \Diamond F_i^{\mathcal{A}} \\ Acc_{\mathcal{B}} &= \bigvee_{1 \le i \le m} \Diamond \Box \neg E_i^{\mathcal{B}} \wedge \Box \Diamond F_i^{\mathcal{B}} \end{aligned}$$

are the Rabin acceptance conditions of $\mathcal{A}$ and $\mathcal{B}$, respectively, then we define the objective for player 0 in the game $\mathcal{G}_{\mathcal{A},\mathcal{B}}$ as the Rabin acceptance condition as

$$\psi = \bigvee_{1 \le i \le n+m} \Diamond \Box \neg E_i \wedge \Box \Diamond F_i$$

where for $1 \le i \le n$ and $1 \le j \le m$:

$$\begin{aligned} E_i &= \{\langle q, p \rangle \in V_0 \mid q \in E_i^{\mathcal{A}}\} & \qquad E_{n+j} &= \{\langle q, p \rangle \in V_0 \mid p \in E_j^{\mathcal{B}}\} \\ F_i &= \{\langle q, p \rangle \in V_0 \mid q \in F_i^{\mathcal{A}}\} & \qquad F_{n+j} &= \{\langle q, p \rangle \in V_0 \mid p \in F_j^{\mathcal{B}}\} \end{aligned}$$

Lemma 5 (Game-based characterization of the GFG property). *$\mathcal{A}$ is good-for-games iff player 0 has a winning strategy in the Rabin game $\mathcal{G}_{\mathcal{A},\mathcal{B}}$.*

In what follows, let $\mathcal{G} = \mathcal{G}_{\mathcal{A},\mathcal{B}}$.

Proof of "$\Longleftarrow$". Assume that player 0 has a winning strategy $\mathfrak{g} : (V_1\, V_0)^+ \to V_1$ in $\mathcal{G}$. To define a GFG-strategy $\mathfrak{f} : (Q \times \Sigma)^+ \to Q$ for player 0 in the monitor game for $\mathcal{A}$, we first look at the play fragment $q_0\, \sigma_1\, q_1\, \sigma_2\, \ldots\, q_n\, \sigma_n$ and consider the choice of $\mathfrak{g}$ in $\mathcal{G}$ for the following play fragment in $\mathcal{G}$:

$$\varsigma \;=\; \langle q_0, p_0\rangle\, \langle q_0, p_0, \sigma_1\rangle\, \langle q_1, p_1\rangle\, \langle q_1, p_1, \sigma_2\rangle\, \ldots\, \langle q_n, p_n\rangle\, \langle q_n, p_n, \sigma_n\rangle$$

where $p_i = \delta_{\mathcal{B}}(p_0, \sigma_0\, \sigma_1 \ldots \sigma_{i-1})$ is the unique state in $\mathcal{B}$ that is reached from p_0 by reading the finite input string $\sigma_0\, \sigma_1 \ldots \sigma_{i-1}$. Let $\langle q_{n+1}, p_{n+1}\rangle = \mathfrak{g}(\varsigma)$. Then, we define:

$$\mathfrak{f}(q_0\, \sigma_1\, q_2\, \sigma_2\, \ldots\, q_n\, \sigma_n) \;=\; q_{n+1}$$

Suppose that $w = \sigma_0\, \sigma_1\, \sigma_2\, \ldots \in \mathcal{L}(\mathcal{A})$. Since the strategy $\mathfrak{g}$ is winning, the $\mathfrak{g}$-play induced by w,

$$\varsigma \;=\; \langle q_0, p_0\rangle\, \langle q_0, p_0, \sigma_1\rangle\, \langle q_1, p_1\rangle\, \langle q_1, p_1, \sigma_2\rangle\, \ldots,$$

in $\mathcal{G}$ is winning for player 0, i.e., ς satisfies the Rabin condition ψ associated with $\mathcal{G}$. We pick some Rabin pair (E_i, F_i) such that $\varsigma \models \Diamond\Box\neg E_i$ and $\varsigma \models \Box\Diamond F_i$. By taking the projection of all states in (E_i, F_i) we obtain a Rabin pair $(E_i^{\mathcal{A}}, F_i^{\mathcal{A}})$ in $\mathcal{A}$ and the $\mathfrak{f}$-play induced by w is $q_0\, \sigma_1\, q_1\, \sigma_2 \ldots$. Hence, $\varsigma|_{\mathcal{A}} \;=\; q_0\, q_1\, \ldots$ is a run for w in $\mathcal{A}$ and

$$\varsigma|_{\mathcal{A}} \models \Diamond\Box\neg E_i^{\mathcal{A}} \quad \text{and} \quad \varsigma|_{\mathcal{B}} \models \Box\Diamond F_i^{\mathcal{A}}.$$

Thus, ς meets the Rabin condition of $\mathcal{A}$.

Proof of "$\Longrightarrow$". Assume $\mathcal{A}$ is good-for-games. Then, there exists a GFG-strategy $\mathfrak{f} : (Q \times \Sigma)^+ \to Q$ for the monitor game for $\mathcal{A}$. We define the strategy $\mathfrak{g}$ for player 0 in $\mathcal{G}$ as follows. Given the play fragment

$$\varsigma \;=\; \langle q_0, p_0\rangle\, \langle q_0, p_0, \sigma_1\rangle\, \langle q_1, p_1\rangle\, \langle q_1, p_1, \sigma_2\rangle\, \ldots\, \langle q_n, p_n\rangle\, \langle q_n, p_n, \sigma_n\rangle$$

in $\mathcal{G}$ we define $\mathfrak{g}(\varsigma)$ as follows:

$$\mathfrak{g}(\varsigma) \;=\; \langle \mathfrak{f}(q_0\, \sigma_1\, q_1\, \sigma_2\, \ldots q_n\, \sigma_n), \delta_{\mathcal{B}}(p_n, \sigma_n)\rangle$$

Let $w = \sigma_0\, \sigma_1\, \sigma_2 \ldots \in \Sigma^\omega$ be an infinite word.
Case 1: $w \in \mathcal{L}(\mathcal{A})$. Then, for the $\mathfrak{g}$-play $\varsigma \;=\; \langle q_0, p_0\rangle\, \langle q_0, p_0, \sigma_0\rangle\, \langle q_1, p_1\rangle\, \langle q_1, p_1, \sigma_1\rangle\, \ldots$ induced by w in $\mathcal{G}$ we have:

$$\varsigma|_{\mathcal{A}} \;=\; q_0\, q_1\, q_2\, \ldots \;\models\; \Diamond\Box\neg E_i^{\mathcal{A}} \;\wedge\; \Box\Diamond F_i^{\mathcal{A}}$$

for some $i \in \{1, \ldots, n\}$. As $\mathcal{G}$'s objective ψ contains the corresponding Rabin pair (E_i, F_i), we get $\varsigma \models \psi$.

Case 2: $w \notin \mathcal{L}(\mathcal{A})$. Then $w \in \mathcal{L}(\mathcal{B})$. Let ρ be the unique run for w in $\mathcal{B}$. Then, $\rho \models Acc_{\mathcal{B}}$. Hence, there exists $j \in \{1, \ldots, m\}$ with $\rho \models \Diamond\Box\neg E_j^{\mathcal{B}}$ and $\rho \models \Box\Diamond F_j^{\mathcal{B}}$. Let

$$\varsigma \;=\; \langle q_0, p_0\rangle\,\langle q_0, p_0, \sigma_0\rangle\,\langle q_1, p_1\rangle\,\langle q_1, p_1, \sigma_1\rangle\,\langle q_2, p_2\rangle\,\langle q_2, p_2, \sigma_2\rangle\;\ldots$$

be the $\mathfrak{g}$-play in $\mathcal{G}$ if player 1 chooses the symbols σ_i according to w. Then, $\varsigma|_{\mathcal{B}} = \rho$. Hence, $\varsigma \models \Diamond\Box\neg E_{n+j}$ and $\varsigma \models \Box\Diamond F_{n+j}$. Thus, ς satisfies the objective ψ for player 0 in $\mathcal{G}$. $\Box$

In practice, when starting from an LTL formula φ, we do not construct $\mathcal{B}$ from $\mathcal{A}$ via complementation and determinization. As we know that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\varphi)$, we obtain the DRA $\mathcal{B}$ by applying standard algorithms for the construction of a DRA for the negation $\neg\varphi$, i.e., via the tool LTL2DSTAR. Even though $\mathcal{A}$ and $\mathcal{B}$ (and thus the game as well) have a worst case double exponential number of states in the size of the formula, and though solving Rabin games is itself NP-complete, we have been able in practice for the smaller automata (see Section 5) to use this approach to determine whether the intermediate automata in the iterative approach of the HP-construction are GFG or not.

F Union Operator for Nondeterministic Rabin GFG Automata

The idea for the union optimisation is inspired from [14]. For constructing a deterministic Rabin automaton for a LTL formula $\varphi = \varphi_1 \vee \varphi_2$ the union optimisation in [14] constructs a DRA $\mathcal{A}_1$ for φ_1 and an DRA $\mathcal{A}_2$ for φ_2. A union operator is applied to $\mathcal{A}_1$ and $\mathcal{A}_2$ yielding an automaton $\mathcal{A}_1 \cup \mathcal{A}_2$ with $\mathcal{L}(\mathcal{A}_1 \cup \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. We now show that this approach is also applicable for GFG automata with Rabin acceptance.

Definition 6 (Union of two NRA). *Let $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, Acc_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, Acc_2)$ be two complete NRA over the same alphabet. The NRA $\mathcal{A}_1 \cup \mathcal{A}_2 = (Q', \Sigma, \delta', q_0', Acc')$ is defined as follows. The state space of $\mathcal{A}_1 \cup \mathcal{A}_2$ is $Q' = Q_1 \times Q_2$ and $q_0' = (q_{0,1}, q_{0,2})$ its initial state. The transition function δ' is given by:*

$$\delta'((q_1, q_2), \sigma) \;=\; \big\{\,(q_1', q_2') : q_1' \in \delta_1(q_1, \sigma),\, q_2' \in \delta_2(q_2, \sigma)\,\big\}$$

The acceptance condition Acc' is given by:

$$\bigvee_{(E,F)\in Acc_1} \big(\Diamond\Box\neg(E \times Q_2) \wedge \Box\Diamond(F \times Q_2)\big) \;\vee\; \bigvee_{(E,F)\in Acc_2} \big(\Diamond\Box\neg(Q_1 \times E) \wedge \Box\Diamond(Q_1 \times F)\big)$$

Obviously, $\mathcal{L}(\mathcal{A}_1 \cup \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. Additionally, the union operation preserves the GFG property.

Lemma 7 (Good-for-games for the union operation). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be complete NRA. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are GFG, then $\mathcal{A}_1 \cup \mathcal{A}_2$ is good-for-games, too.*

Proof. Let $\mathfrak{f}_1 : (Q_1 \times \Sigma)^+ \to Q_1$ be a GFG-strategy for $\mathcal{A}_1$ and $\mathfrak{f}_2 : (Q_2 \times \Sigma)^+ \to Q_2$ be a GFG-strategy for $\mathcal{A}_2$. We define a strategy

$$\mathfrak{f} : (Q' \times \Sigma)^+ \to Q'$$

for $\mathcal{A}_1 \cup \mathcal{A}_2$ as follows:

$$\mathfrak{f}(\varsigma \downarrow i) \ \overset{\text{def}}{=} \ \langle\, \mathfrak{f}_1(\varsigma\big|_1 \downarrow i),\, \mathfrak{f}_2(\varsigma\big|_2 \downarrow i)\,\rangle$$

where $\varsigma\big|_1 = q_{0,1}\,\sigma_0\,q_{1,1}\,\ldots q_{i-1,1}\sigma_i$ denotes the projection of the play

$$\varsigma \ = \ \langle q_{0,1}, q_{0,2}\rangle\,\sigma_0\,\langle q_{1,1}, q_{1,2}\rangle\,\sigma_1\,\ldots\,\langle q_{i,1}, q_{i,2}\rangle\,\sigma_i$$

to the first automaton. $\varsigma\big|_2$ is defined analogously.

The goal is to show that $\mathfrak{f}$ is a GFG-strategy for $\mathcal{A}_1 \cup \mathcal{A}_2$. Let $w = \sigma_0\,\sigma_1\,\sigma_2\,\ldots \in \mathcal{L}(\mathcal{A}_1 \cup \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$ and let

$$\varsigma \ = \ \langle q_{0,1}, q_{0,2}\rangle\,\sigma_0\,\langle q_{1,1}, q_{1,2}\rangle\,\sigma_1\,\langle q_{2,1}, q_{2,2}\rangle\,\sigma_2\ldots$$

be the induced $\mathfrak{f}$-play in the monitor game for $\mathcal{A}_1 \cup \mathcal{A}_2$ with $\varsigma\big|_\Sigma = w$. W.l.o.g. we may suppose that $w \in \mathcal{L}(\mathcal{A}_1)$. By the definition of $\mathfrak{f}$, the play $\varsigma\big|_1 = q_{0,1}\,\sigma_0\,q_{1,1}\,\sigma_1\,q_{2,1}\,\sigma_2\ldots$ in the monitor game for $\mathcal{A}_1$ is $\mathfrak{f}_1$-conform, and hence $q_{0,1}\,q_{1,1}\,q_{2,1}\ldots$ is an accepting run in $\mathcal{A}_1$. Thus, there is a Rabin pair $(E_w, F_w) \in Acc_1$ with

$$q_{0,1}\,q_{1,1}\,\ldots \models \Diamond\Box\neg E_w \wedge \Box\Diamond F_w$$

Hence, for the run $\varsigma\big|_Q = \langle q_{0,1}, q_{0,2}\rangle\,\langle q_{1,1}, q_{1,2}\rangle\,\langle q_{2,1}, q_{2,2}\rangle\ldots$ in $\mathcal{A}_1 \cup \mathcal{A}_2$ we have:

$$\varsigma\big|_Q \models \Diamond\Box\neg(E_w \times Q_2) \ \wedge \ \Box\Diamond(F_w \times Q_2)$$

We conclude that $\varsigma\big|_Q$ is an accepting run for the word w in $\mathcal{A}_1 \cup \mathcal{A}_2$.

G Implementation and Experiments

G.1 Statistics for Selected Formulas

We provide here some more statistics for selected formulas to allow an individual comparison between the GFG automata generated with LTL2GFG and the DRA generated by LTL2DSTAR.

Table 3 lists the size of the constructed automata in terms of the number of reachable states. We consider here the standard variant of the HP-algorithm [13] and combinations of the loose, the union and the iterative variant. We contrast this with the results of LTL2DSTAR in the default variant. Missing entries correspond to timeouts during the generation. Table 4 lists the corresponding size of the automata in terms of BDD nodes used for encoding the transition function of the automata. The tables were generated with dynamic reordering of the variable order activated.

In Table 4, we see as well one of the formula where the BDD size of the GFG automaton was (marginally) better than the BDD size of the DRA obtained by LTL2DSTAR. For $\Box a$ the BDD for the transition function consists of 8 nodes in LTL2GFG, while the BDD of LTL2DSTAR consists of 10 nodes.

Table 3. Detailed statistics for example formulas: number of reachable states

Formula	ltl2gfg					ltl2dstar
	standard	loose	union, loose	iter., loose	union, it., loose	
true	3	3	3	3	3	2
false	2	2	2	2	2	1
a	6	8	8	4	4	3
$\neg a$	6	8	8	4	4	3
$a \wedge b$	6	8	8	4	4	3
$a \vee b$	6	8	50	4	10	3
$a \rightarrow b$	6	8	50	4	10	3
$\Box\, a$	3	3	3	3	3	3
$\Diamond\, a$	16	46	46	6	6	2
$\Box \Diamond\, a$	33	73	73	9	9	2
$\Diamond \Box\, a$	16	46	46	46	46	2
$\Diamond \Box\, a \rightarrow \Box \Diamond\, b$	–	$3.3 \cdot 10^9$	5329	6482	81	2
$\Box \Diamond\, a \rightarrow \Box \Diamond\, b$	–	$2.2 \cdot 10^9$	3358	–	414	4
$a \mathcal{U}\, b$	16	46	46	6	6	3
$a \mathcal{U} (b \mathcal{U}\, c)$	224	7734	7734	12	12	4
$\Box(a \rightarrow \Diamond\, b)$	33	73	73	9	9	4

G.2 Implementation and Experiments with Prism

We provide here some further details on our implementation of the GFG-based probabilistic model checking approach in Prism. We construct a BDD-based representation of the GFG automaton with ltl2gfg, which we import into the MTBDD engine of Prism. We consider the GFG automata as Rabin automata, as Prism currently has no specialized handling of automata with the parity acceptance condition. To perform the product construction with $\mathcal{M}$, we have to represent the nondeterministic choice of the successor state in the GFG automaton symbolically, which requires one additional copy of each boolean state variable. Taken together with the action variables in the MDP $\mathcal{M}$, they then represent the corresponding action in the product. For the fully symbolic engine of Prism, these action variables can remain interleaved. The hybrid engine (symbolic matrix and explicit value vector) however requires that all action variables appear on top of the variable ordering, before any state variables. Our experiments indicated that this encoding leads to a significant blow-up of the symbolic representation. Thus, we focus here only on the results using the fully symbolic engine.

GFG-based analysis of MDPs. For our experiments, we used a Prism model [23] from the PRISM benchmark suite for parts of the WLAN carrier-sense protocol of IEEE 802.11. For details on the model we refer to
`http://www.prismmodelchecker.org/casestudies/wlan.php`

It models a two-way handshake mechanism of the IEEE 802.11 (WLAN) medium access control scheme with two senders that compete for the medium. As messages get corrupted when both senders send at the same time (called a collision), a probabilistic back-off mechanism is employed to reduce the likelihood

Table 4. Detailed statistics for example formulas: size of the transition function BDD

Formula	LTL2GFG					LTL2DSTAR
	standard	loose	union, loose	iter., loose	union, it., loose	
$true$	7	7	7	7	7	3
$false$	7	7	7	7	7	1
a	59	46	46	17	17	11
$\neg a$	59	46	46	17	17	11
$a \wedge b$	78	48	48	18	18	13
$a \vee b$	78	48	90	18	32	13
$a \rightarrow b$	78	48	90	18	32	13
$\Box a$	8	8	8	8	8	10
$\Diamond a$	123	85	85	22	22	6
$\Box \Diamond a$	136	70	70	27	27	5
$\Diamond \Box a$	141	70	70	76	76	5
$\Diamond \Box a \rightarrow \Box \Diamond b$	–	69669	147	2383	51	6
$\Box \Diamond a \rightarrow \Box \Diamond b$	–	12710	149	–	107	8
$a \mathcal{U} b$	132	102	102	23	23	15
$a \mathcal{U} (b \mathcal{U} c)$	5165	1642	1642	62	62	21
$\Box(a \rightarrow \Diamond b)$	182	97	97	31	31	13

of collisions. The key feature of the protocol is a back-off procedure, which is started if an error occurred or the sender wants to send a new message after sending a message. The back-off procedure consists of waiting a randomized amount of time while the channel has to be free. It ends with retrying to send a message. To define the maximal amount of waiting time in the back-off procedure, the model is parametrized by the parameter `MAX-BACKOFF`. Here, we consider the values `MAX-BACKOFF` $\in \{1, \ldots, 6\}$. Since stations cannot listen to their own transmissions, after having started to transmit a message, they cannot determine for a short amount of time whether another station has started to send at the same moment, called the *vulnerable* section. To reduce the likelihood of this collisions, the station has to check that the channel is free for a fixed time period. This happens in state `Wait_Difs`. In the model, a collision counter is used to record the number of collisions for use in the formulas. For our experiments, we set the maximum number of collisions that can be counted to 4.

LTL Formulas. We report here on our results when considering the following LTL path properties:

- $\varphi_1 = \Diamond(s_1 = \texttt{Done} \wedge s_2 = \texttt{Done})$
  "Eventually station 1 and station 2 have sent their message correctly"
- $\varphi_2 = \neg \Diamond \Box(s_1 = \texttt{Backoff} \vee s_1 = \texttt{Wait\_Difs} \vee s_1 = \texttt{Wait\_ack})$
  "Station 1 never never gets stuck in the back-off procedure or during one of the waiting procedures"

- $\varphi_3 = (\Box \Diamond s_1 = \texttt{Backoff}) \rightarrow (\Box \Diamond s_1 = \texttt{vuln})$
  "If station 1 backs off infinitely often, it is also infinitely often in its vulnerable section"

- $\varphi_4 = \Box(\texttt{col} \rightarrow \Diamond \texttt{msg\_1\_send})$
  "If a collision occurred on the channel, then station 1 will nevertheless send its message correctly at some point in the future"

- $\varphi_5 = \Box(\texttt{col} \rightarrow \Diamond(\texttt{msg\_1\_send} \wedge \texttt{\#col} < 2))$
  "Like φ_4, but with the additional constraint of no more additional collisions"

- $\varphi_6 = \Diamond(s_1 = \texttt{Done} \wedge s_2 = \texttt{Done}) \quad \wedge \quad \Box(\texttt{\# col} < 2)$
  "Eventually both stations have sent their messages and the number of collisions never exceeds 1"

Experiments with PRISM. We report here on a comparison of the model checking times for calculating the probabilities $\Pr^{\min}_{\mathcal{M}}(\varphi)$ for the formulas above with PRISM, once using the GFG implementation and once with the standard settings. For φ_1, which is normally handled by PRISM via a specialized algorithm for simple formulas that does not need an automata product construction, we forced the use of the general, automata-based approach. As explained above, we used the fully symbolic MTBDD engine of PRISM, i.e., where the matrix and the value vector are stored symbolically.

We have carried out our experiments with the different variants for the generation of GFG automata of LTL2GFG. As before, we impose a 30 minute time and 10GB memory limit. In our tables, we will refer by `WLAN`n to the case-study MDP with maximum back-off value `MAX_BACKOFF`, for n from 1 to 6. Table 5 lists the time spent for calculating $\Pr^{\min}_{\mathcal{M}}(\varphi)$ for each of the six LTL formulas φ and the `WLAN4` MDP. For this table, we list the results using the optimal variant of LTL2GFG for each formula. In all cases in this table, the time spent generating the GFG automata was less than one second. This allows a fair comparison against the standard PRISM approach using DRA. Interestingly, the BDD sizes listed in Table 5 for the number of MTBDD nodes used to encode the transition matrix of the product $\mathcal{M} \otimes \mathcal{A}$ is roughly similar between the standard approach based on DRA and the best LTL2GFG approach. However, the time then spent for calculating the probabilities in the product are vastly higher for the GFG approach than for the DRA approach.

Table 6 lists the time spent in model checking the different MDPs and formulas in PRISM using the standard approach. Table 7 lists the corresponding values for the approach using LTL2GFG in the loose variant, which also employed the iterative and union approach. This variant generally behaved well in these experiments. We list as well the time spent for constructing the GFG NRA for $\neg\varphi_i$. The formula is negated because we are interesting in the minimal probabilities.

To provide an overview of the behavior of the different variants of LTL2GFG in the context of PRISM, Table 8 compares the running time of some of the variants against the baseline of the DRA-based standard approach. We consider the 36

Table 5. Results for model checking `WLAN4`

	LTL2GFG		PRISM standard	
	time	BDD size $\mathcal{M} \otimes \mathcal{A}$	time	BDD size $\mathcal{M} \otimes \mathcal{A}$
$\Pr^{\min}_{\mathcal{M}}(\varphi_1)$	23.4 s	25,587	5.7 s	25,725
$\Pr^{\min}_{\mathcal{M}}(\varphi_2)$	38.4 s	29,184	0.7 s	27,518
$\Pr^{\min}_{\mathcal{M}}(\varphi_3)$	63.4 s	39,493	2.9 s	30,355
$\Pr^{\min}_{\mathcal{M}}(\varphi_4)$	43.6 s	27,811	3.1 s	26,043
$\Pr^{\min}_{\mathcal{M}}(\varphi_5)$	43.5 s	27,573	3.1 s	26,043
$\Pr^{\min}_{\mathcal{M}}(\varphi_6)$	97.6 s	26,002	42.9 s	25,849

Table 6. Time consumption for model checking with standard PRISM

	$\Pr^{\min}_{\mathcal{M}}(\varphi_1)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_2)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_3)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_4)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_5)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_6)$
`WLAN1`	0.4 s	0.1 s	0.2 s	0.2 s	0.2 s	1.9 s
`WLAN2`	0.8 s	0.2 s	0.5 s	0.4 s	0.4 s	5.7 s
`WLAN3`	1.4 s	0.3 s	1.0 s	1.1 s	1.1 s	14.2 s
`WLAN4`	5.4 s	0.5 s	2.4 s	2.5 s	2.8 s	42.5 s
`WLAN5`	17.2 s	1.0 s	5.8 s	6.4 s	7.4 s	132.3 s
`WLAN6`	50.8 s	2.2 s	16.0 s	20.5 s	21.6 s	556.5 s

Table 7. Time consumption for PRISM and LTL2GFG (variant loose, iterative and union)

	$\Pr^{\min}_{\mathcal{M}}(\varphi_1)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_2)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_3)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_4)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_5)$	$\Pr^{\min}_{\mathcal{M}}(\varphi_6)$
Constructing GFG $\mathcal{A}_{\neg\varphi_i}$	0.3 s	0.3 s	0.5 s	0.3 s	0.3 s	0.4 s
`WLAN1`	1.4 s	2.2 s	4.8 s	3.7 s	2.6 s	3.7 s
`WLAN2`	2.8 s	5.1 s	9.9 s	7.9 s	5.2 s	9.9 s
`WLAN3`	6.8 s	11.7 s	22.8 s	19.5 s	13.7 s	29.3 s
`WLAN4`	23.8 s	40.2 s	68.8 s	55.5 s	45.6 s	100.6 s
`WLAN5`	96.2 s	148.8 s	249.8 s	175.2 s	171.6 s	369.4 s
`WLAN6`	392.2 s	603.0 s	1133.3 s	729.2 s	698.1 s	1510.1 s

combinations of `WLAN1` to `WLAN6` and φ_1 to φ_6. The table lists the number of cases where a timeout occured and where time spent using the GFG approach exceeded the standard approach only by a given factor. We refer to the baseline time spent using the standard approach in Prism as t_{STD} and to the time spent using the GFG approach (when there was no timeout) as t_{GFG}. For example,

Table 8. Results for model checking with Prism and different variants of LTL2GFG

	std.	loose	dyn.,loose	un.,loose	it.,loose	it.,loose,dyn.	it.,un.,loose
$t_{\mathrm{GFG}} < 3 \cdot t_{\mathrm{STD}}$	0	1	1	0	0	0	5
$t_{\mathrm{GFG}} < 7 \cdot t_{\mathrm{STD}}$	5	5	5	10	9	7	11
$t_{\mathrm{GFG}} < 20 \cdot t_{\mathrm{STD}}$	15	15	14	19	21	18	22
$t_{\mathrm{GFG}} < 250 \cdot t_{\mathrm{STD}}$	33	29	33	34	35	32	35
$t_{\mathrm{GFG}} \leq 30\,\mathtt{min}$	34	31	33	35	35	35	36
Aborted	2	5	3	1	1	1	0

if we consider the loose variant with active iterative approach, in 9 of the 36 cases the run time of Prism with the GFG approach was within the time spent by the standard Prism approach multiplied by the factor 7. As can be seen, the loose variant with union and the iterative approach fared well in general. Interestingly, the automata generated with active dynamic reordering in some cases fared significantly worse than those using the initial variable ordering. As Prism does not support a reordering of the variables, the BDD representation of the GFG automaton is optimized by the dynamic reordering in LTL2GFG for the stand-alone representation. Clearly, this variable ordering may however not be optimal for the product with the MDP. Likewise, the dynamic variable reordering sometimes slowed down the GFG check in the iterative approach.

As we have seen, the GFG approach implemented in Prism did not improve on the model checking time of the standard, explicit determinization based approach, even for the relatively simple LTL formulas used in our experiments.