Good-for-MDP State Reduction for Stochastic LTL Planning

Christoph Weinhuber[1], **Giuseppe De Giacomo**[1], **Yong Li**[2*], **Sven Schewe**[3], **Qiyi Tang**[3]
[1]University of Oxford, Oxford, UK
[2]Key Laboratory of System Software (Chinese Academy of Sciences),
Institute of Software Chinese Academy of Sciences, PRC
[3]University of Liverpool, UK
{christoph.weinhuber, giuseppe.degiacomo}@cs.ox.ac.uk, liyong@ios.ac.cn, {sven.schewe, qiyi.tang}@liverpool.ac.uk

Abstract

We study stochastic planning problems in Markov Decision Processes (MDPs) with goals specified in Linear Temporal Logic (LTL). The state-of-the-art approach transforms LTL formulas into good-for-MDP (GFM) automata, which feature a restricted form of nondeterminism. These automata are then composed with the MDP, allowing the agent to resolve the nondeterminism during policy synthesis. A major factor affecting the scalability of this approach is the size of the generated automata. In this paper, we propose a novel GFM state-space reduction technique that significantly reduces the number of automata states. Our method employs a sophisticated chain of transformations, leveraging recent advances in good-for-games minimisation developed for adversarial settings. In addition to our theoretical contributions, we present empirical results demonstrating the practical effectiveness of our state-reduction technique. Furthermore, we introduce a direct construction method for formulas of the form $\mathsf{GF}\varphi$, where φ is a co-safety formula. This construction is provably single-exponential in the worst case, in contrast to the general doubly-exponential complexity. Our experiments confirm the scalability advantages of this specialised construction.

1 Introduction

Planning with temporal objectives has a long tradition in artificial intelligence. Early pioneering work, such as (Bacchus, Boutilier, and Grove 1996, 1997; Littman 1997; Littman, Goldsmith, and Mundhenk 1998; Thiébaux et al. 2006), defined key formalisms like *Markov decision processes* (MDPs), analysed the computational complexity of planning problems and introduced methods for handling *non-Markovian rewards*.

These foundations paved the way for more expressive and structured approaches, most notably the use of *Linear Temporal Logic* (LTL) (Pnueli 1977), to formally specify complex goals (Lacerda, Parker, and Hawes 2014; Brafman and Giacomo 2024). This field is now even pushing into areas like reinforcement learning (Yang, Littman, and Carbin 2022), multi-agent systems (Schillinger, Bürger, and Dimarogonas 2019), and handling uncertainty (Yu et al. 2025).

Traditionally, in order to solve an MDP with an LTL goal φ, we first build a *nondeterministic Büchi automaton* (NBA) for φ. Then we convert the NBA to an equivalent *deterministic automaton*, e.g., a deterministic Rabin automaton (DRA) (Safra 1988). Recent progresses in this direction include direct translations of LTL to DRAs (Esparza and Kretínský 2014). Afterwards, we need to perform the Cartesian product of the MDP and the DRA, identify *accepting end-components* (ECs) and compute a *(memoryless) strategy* with maximal probability of reaching accepting ECs.

State-of-the-art approaches (Hahn et al. 2015; Sickert et al. 2016) proposed the usage of *limit-deterministic Büchi automata* (LDBAs) to avoid the notorious problem of obtaining DRAs. LDBAs give the burden of resolving the *nondeterminism* in automata to the agent, without changing the optimal satisfaction probability. Yet, only a certain type of LDBAs works for all finite MDPs. The precise criterion for NBAs is expressed by being *"good-for-MDP"* (GFM), where all nondeterministic choices are *angelic* in the sense that they can be resolved by the agent, while preserving the optimal satisfaction probability (Hahn et al. 2020). While the LTL to GFM construction does not change the complexity of the overall planning problem, in practice, the difference among using GFMs and DRAs is significant (Meyer, Sickert, and Luttenberger 2018).

A major factor affecting the scalability of solving MDPs with LTL goals is the size of the GFM automata corresponding to the LTL goals (Hahn et al. 2015; Sickert and Kretínský 2016). In this paper, we propose a novel GFM state-space reduction technique that significantly reduces the number of automata states. Our method employs a sophisticated chain of transformations, which allows us to leverage recent advances in *good-for-games* minimisation developed for adversarial settings (Radi and Kupferman 2022) to our context. We provide experimental evidence that our state-reduction technique effectively reduces the automata state space, by taking benchmark examples of LTL goals from eight sources, including influential papers in planning, verification and reinforcement learning.

Furthermore, we introduce a *direct* method for constructing the GFM automata of formulas of the form $\mathsf{GF}\varphi$, where φ is a *co-safety* property. This kind of formulas is common in both verification (Holeček et al. 2004) and reinforcement learning (Jackermeier and Abate 2025). We show that the GFM automaton obtained through our specialised construction incurs only a *singly* exponential blow-up, in contrast to

*Corresponding author
Preprint. Accepted to AAAI 2026.

the doubly exponential blow-up in the general case. Moreover, we provide experimental evidence that this theoretical advantage indeed translates into significantly fewer states.

2 Preliminaries

Markov Decision Processes. Following (Baier and Katoen 2008), a Markov decision process (MDP) $\mathcal{M}$ is a tuple $(S, \mathrm{Act}, \Sigma, \mathsf{P}, s_0, L)$ with a finite set of states S, a set of actions Act, a set of labels Σ, a transition probability function $\mathsf{P} : S \times \mathrm{Act} \times S \to [0, 1]$, an initial state $s_0 \in S$ and a labelling function $L : S \times \mathrm{Act} \to \Sigma$ that labels state-action pairs to the set of propositions that hold in that state[1]. We abuse the notation by writing $\mathrm{Act}(s)$ to denote the set of available actions at state s. A path ξ of $\mathcal{M}$ is an (in)finite sequence of alternating states and actions $\xi = s_0 a_0 s_1 a_1 \cdots$, ending with a state if finite, such that for all $i \geq 0$, $a_i \in \mathrm{Act}(s_i)$ and $\mathsf{P}(s_i, a_i, s_{i+1}) > 0$. The sequence $L(\xi) = L(s_0, a_0)L(s_1, a_1), \cdots$ over Σ is called the *trace* induced by the path ξ over $\mathcal{M}$. We denote by FPaths and IPaths the set of all finite and infinite paths of $\mathcal{M}$.

A (finite-memory) strategy μ of $\mathcal{M}$ is a function $\mu : \mathrm{FPaths} \to \mathsf{Distr}(\mathrm{Act})$ such that, for each $\xi \in \mathrm{FPaths}$, $\mu(\xi) \in \mathsf{Distr}(\mathrm{Act}(\mathsf{lst}(\xi)))$, where $\mathsf{lst}(\xi)$ is the last state of the finite path ξ and $\mathsf{Distr}(\mathrm{Act})$ denotes the set of all possible distributions over Act. Let $\Omega^{\mathcal{M}}_{\mu}(s_0)$ denote the subset of (in)finite paths of $\mathcal{M}$ that correspond to strategy μ and initial state s_0. μ is *memoryless* if $\mu : S \to \mathrm{Distr}(\mathrm{Act})$, meaning it selects actions independently of the past.

A strategy μ of $\mathcal{M}$ is able to resolve the nondeterminism of an MDP and induces a Markov chain (MC) $\mathcal{M}^{\mu} = (\mathrm{FPaths}, \Sigma, \mathsf{P}_{\mu}, L')$ where, for $\xi = s_0 a_0 \cdots s_{n-1} a_{n-1} s_n \in \mathrm{FPaths}$ and $a_n \in \mathrm{Act}(s_n)$, $\mathsf{P}_{\mu}(\xi, \xi \cdot a_n \cdot s_{n+1}) = \mathsf{P}(s_n, a_n, s_{n+1}) \cdot \mu(\xi)(a_n)$ and $L'(\xi) = L(s_{n-1}, a_{n-1})$.

A *sub-MDP* of $\mathcal{M}$ is an MDP $\mathcal{M}' = (S', \mathrm{Act}', \Sigma, \mathsf{P}', L)$ where $S' \subseteq S, \mathrm{Act}' \subseteq \mathrm{Act}$ is such that, for every $s \in S'$, $\mathrm{Act}'(s) \subseteq \mathrm{Act}(s)$, and P' and L' are analogous to P and L when restricted to S' and Act'. In particular, $\mathcal{M}'$ is closed under probabilistic transitions, i.e., for all $s \in S'$ and $a \in \mathrm{Act}'$ we have that $\mathsf{P}(s, a, s') > 0$ implies $\mathsf{P}'(s, a, s') > 0$. An *end-component* (EC) of an MDP $\mathcal{M}$ is a sub-MDP $\mathcal{M}'$ of $\mathcal{M}$ such that its underlying graph is strongly connected. A maximal end-component (MEC) is an EC $\mathcal{E} = (E, \mathrm{Act}', \Sigma, \mathsf{P}', L)$ such that there is no other EC $\mathcal{E} = (E', \mathrm{Act}'', \Sigma, \mathsf{P}'', L)$ such that $E \subset E'$. An MEC $\mathcal{E}$ that cannot reach states outside $\mathcal{E}$ is a *leaf* component.

Theorem 1 ((de Alfaro 1997; Baier and Katoen 2008))**.** *Once an EC $\mathcal{E}$ of an MDP is entered, there is a strategy that visits every state-action combination in $\mathcal{E}$ with probability* 1 *and stays in $\mathcal{E}$ forever. Moreover, for every strategy the union of the end-components is visited with probability* 1*. An infinite path of an MC $\mathcal{M}$ almost surely (with probability* 1*) will enter a leaf component.*

Linear Temporal Logic. An LTL formula, over a finite set of atomic propositions AP is defined as $\varphi ::= a \in \mathsf{AP} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \mathsf{G}\varphi \mid \mathsf{F}\varphi \mid \varphi \mathsf{U} \varphi$. Here X (Next), G (Globally/Always), F (Finally/Eventually) and U (Until) are temporal operators.

An ω-trace w is an infinite sequence of letters $w[0]\, w[1]\, w[2] \ldots$ with $w[i] \in \Sigma = 2^{\mathsf{AP}}$. We denote the infinite suffix $w[i]\, w[i+1] \ldots$ by w_i. The satisfaction relation $\models$ between ω-traces w and formulas φ is defined as follows:

$$\begin{array}{lll}
w \models a & \iff & a \in w[0], \\
w \models \neg\varphi & \iff & w \not\models \varphi, \\
w \models \mathsf{X}\varphi & \iff & w_1 \models \varphi, \\
w \models \mathsf{G}\varphi & \iff & \forall k.\ w_k \models \varphi, \\
w \models \mathsf{F}\varphi & \iff & \exists k.\ w_k \models \varphi, \\
w \models \varphi \mathsf{U} \psi & \iff & \exists k. \big(w_k \models \psi \wedge \forall j < k,\ w_j \models \varphi\big)
\end{array}$$

Further, $\forall w. w \models tt$ and $\forall w. w \not\models f\!f$. We denote by $[\varphi]$ the set of *infinite traces* satisfying the LTL formula φ.

Automata. A (nondeterministic) transition system (TS) is a tuple $\mathcal{T} = (Q, q_0, \delta)$, with a finite set of states Q, an initial state $q_0 \in Q$, and a transition function $\delta : Q \times \Sigma \to 2^Q$. We extend δ to sets by $\delta(S, \sigma) := \bigcup_{q \in S} \delta(q, \sigma)$. A *deterministic* TS satisfies $|\delta(q, \sigma)| \leq 1$ for each $q \in Q$ and $\sigma \in \Sigma$.

An automaton $\mathcal{A}$ is defined as a tuple $(\mathcal{T}, \alpha)$, where $\mathcal{T}$ is a TS and α is an acceptance condition. We define $\Delta(\delta) = \{(q, \sigma, q') \mid q, q' \in Q, \sigma \in \Sigma, q' \in \delta(q, \sigma)\}$. For an infinite trace $w \in \Sigma^{\omega}$, a *run* of $\mathcal{A}$ on w is an infinite sequence of transitions $\rho = (q_0, w[0], q_1)(q_1, w[1], q_2) \cdots$ such that, for every $i \geq 0$, $q_{i+1} \in \delta(q_i, w[i])$. Let $\inf(\rho)$ be the set of transitions that occur infinitely often in a run ρ. The acceptance condition $\alpha \subseteq \Delta(\delta)$ is a set of *accepting* (*rejecting*, resp.) transitions for Büchi (co-Büchi, resp.). A run ρ satisfies the Büchi (co-Büchi, resp.) acceptance condition α if $\inf(\rho) \cap \alpha \neq \emptyset$ ($\inf(\rho) \cap \alpha = \emptyset$, resp.).

A run is *accepting* if it satisfies the condition α; a trace $w \in \Sigma^{\omega}$ is *accepted* by $\mathcal{A}$ if there is an accepting run ρ of $\mathcal{A}$ over w. We use three letter acronyms in $\{D, N\} \times \{B, C\} \times \{A\}$ to denote automata types where the first letter stands for the TS mode, the second for the acceptance type and the third for automaton. For example, DBA means deterministic Büchi automaton. We assume that all automata are *complete*, i.e., for each state $s \in Q$ and letter $\sigma \in \Sigma$, $|\delta(s, \sigma)| \geq 1$.

We denote by $\mathcal{L}(\mathcal{A})$ the *ω-language* recognised by an ω-automaton $\mathcal{A}$, i.e., the set of ω-traces accepted by $\mathcal{A}$.

3 Stochastic Planning Problem

Overview. In a stochastic planning problem, we model the interaction between the agent and the environment as an MDP $\mathcal{M}$ and its task specification as an LTL formula φ. The goal is to find an optimal strategy μ for $\mathcal{M}$ that maximises the chance that the infinite sequence of observed propositions satisfies φ. Formally, we want the probability of the ω-traces generated by $\mathcal{M}^{\mu}$ and belonging to $[\varphi]$, to be maximal among all possible choices of μ.

MDPs and LTL Task Specification. Every LTL formula φ can be translated into a (nondeterministic) Büchi automaton $\mathcal{A}$, accepting $[\varphi]$, i.e., $\mathcal{L}(\mathcal{A}) = [\varphi]$ (Vardi and Wolper 1986). Recall that for a given strategy μ on an MDP $\mathcal{M}$, we can induce an MC $\mathcal{M}^{\mu}$. We define the semantic satisfaction probability of the induced MC $\mathcal{M}^{\mu}$ for $\mathcal{L}(\mathcal{A})$ as

[1]This generalises the usual labelling function $L : S \to \Sigma$.

$\mathbb{P}_{\mathcal{M}^\mu}(\mathcal{L}(\mathcal{A})) = \mathbb{P}\{\xi \in \Omega_\mu^{\mathcal{M}}(s_0) : L(\xi) \in \mathcal{L}(\mathcal{A})\}$, which is the probability of $\mathcal{M}^\mu$ generating an ω-trace in $[\varphi]$. For an MDP $\mathcal{M}$ and an automaton $\mathcal{A}$, we define the *maximal semantic satisfaction probability* as $\mathsf{Psem}(\mathcal{M}, \mathcal{A}) = \sup_\mu \mathbb{P}_{\mathcal{M}^\mu}(\mathcal{L}(\mathcal{A}))$. We look for an *optimal* strategy μ that achieves $\mathsf{Psem}(\mathcal{M}, \mathcal{A})$ in the planning problem.

Product of MDPs and Büchi automata. If $\mathcal{A}$ is a DBA, we can solve the problem of finding an optimal strategy μ by constructing the product MDP $\mathcal{M} \times \mathcal{A}$ and extract a memoryless strategy μ on $\mathcal{M} \times \mathcal{A}$ that reaches accepting components with maximal probability (Baier and Katoen 2008). If $\mathcal{A}$ is nondeterministic, this is in general not possible, because we have to resolve the nondeterminism, which corresponds to future forecasting capabilities that we can assign neither to the agent, nor to the (probabilistic) environment. However, it has been observed (Hahn et al. 2020; Sickert et al. 2016; Hahn et al. 2015) that there is a class of automata called good-for-MDPs (GFM) for which we can assign the nondeterminism to the agent without loss of optimality. Intuitively, a nondeterministic automaton is GFM, if the nondeterminism can be resolved by the agent (we postpone the formal definition of GFM to later in this section). If an automaton is GFM, then we can still adopt a variant of the product construction shown below:

Formally, let $\mathcal{M} = (S, \mathrm{Act}, \Sigma, \mathsf{P}, s_0, L)$ be an MDP and $\mathcal{A} = (Q, q_0, \delta, \alpha)$ be an NBA. We define the product MDP $\mathcal{M} \times \mathcal{A} = (S^\times, \mathrm{Act}^\times, \Sigma, \mathsf{P}^\times, \langle s_0, q_0\rangle, L^\times, \alpha^\times)$ augmented with the acceptance condition $\alpha^\times$ where

- $S^\times = S \times Q$ is the state space.
- $\mathrm{Act}^\times = \mathrm{Act} \times [k]$ is the action set where k is the *maximal* out-degree of $\mathcal{A}$ for any state in Q and letter in Σ. We denote $[k] = \{1, \cdots, k\}$ for any integer $k > 0$.
- $\mathsf{P}^\times : S^\times \times \mathrm{Act}^\times \times S^\times \to [0, 1]$ is the transition probability function such that $\mathsf{P}^\times(\langle s, q\rangle, \langle a, i\rangle, \langle s', q'\rangle) = \mathsf{P}(s, a, s')$ if $\mathsf{P}(s, a, s') > 0$ and $q' \in \delta(q, L(s, a))$ where q' is the i-th state in the *ordered* set $\delta(q, L(s, a))$.
- $L^\times(\langle s, q\rangle, \langle a, i\rangle) = L(s, a)$ for state $\langle s, q\rangle \in S \times Q$ and action $\langle a, i\rangle \in \mathrm{Act}^\times$, and
- For Büchi/co-Büchi, $\alpha^\times = \{(\langle s, q\rangle, \langle a, i\rangle, \langle s', q'\rangle) \mid \mathsf{P}^\times(\langle s, q\rangle, \langle a, i\rangle, \langle s', q'\rangle) > 0, (q, L(s, a), q') \in \alpha\}$.

Instead of storing the selected successor state in the action name as in other literature, e.g. (Hahn et al. 2020), we index the choice by its position in a predefined order over successors. Formally, for each $q \in Q$ and $\sigma \in \Sigma$, we define a function $\mathsf{idx}_{\mathcal{A},q,\sigma}$ mapping each $q' \in \delta(q, \sigma)$ to its unique position $i \in [k]$. This index i serves as the identifier for the agent's choice in our product construction. This encoding is equivalent to (Hahn et al. 2020) but has a slight advantage of using fewer actions. For full details, see Appendix B.

Conditions for GFMness. In order to see if an NBA is GFM, we have to characterise the probabilities that come from the syntactic construction above and show that they are the same of those intrinsic in the original problem. Recall that the probability of the original problem is the following:

$$\mathsf{Psem}(\mathcal{M}, \mathcal{A}) = \sup_\mu \mathbb{P}_{\mathcal{M}^\mu}(\mathcal{L}(\mathcal{A})).$$

The probability coming from the syntactic product construction above is characterised as:

$$\mathsf{Psyn}(\mathcal{M}, \mathcal{A}) = \sup_\mu \mathbb{P}\{\xi \in \Omega_\mu^{\mathcal{M}\times\mathcal{A}}(\langle s_0, q_0\rangle) : \xi \text{ accepting}\}.$$

This can be simplified to $\mathsf{Psyn}(\mathcal{M}, \mathcal{A}) = \sup_\mu \mathbb{P}_{(\mathcal{M}\times\mathcal{A})^\mu}(\mathsf{F}X)$, where X is the set of states of the accepting MECs in $\mathcal{M} \times \mathcal{A}$ that contain accepting transitions in $\alpha^\times$.

Clearly, $\mathsf{Psyn}(\mathcal{M}, \mathcal{A}) \leq \mathsf{Psem}(\mathcal{M}, \mathcal{A})$, because accepting runs ξ only occur on accepting words. Thus, a strategy μ chooses an accepting run on accepting words in the best case, but it is also possible for μ to make wrong decisions.

Obviously, $\mathsf{Psyn}(\mathcal{M}, \mathcal{A}) = \mathsf{Psem}(\mathcal{M}, \mathcal{A})$ if $\mathcal{A}$ is deterministic. Following (Hahn et al. 2020), an NBA $\mathcal{A}$ is said to be GFM if, for all finite MDPs $\mathcal{M}$, $\mathsf{Psem}(\mathcal{M}, \mathcal{A}) = \mathsf{Psyn}(\mathcal{M}, \mathcal{A})$ holds. See Appendix C for a non-GFM case.

LTL to GFM automata. There are several algorithms to transform an LTL formula into an NBA that is GFM (Sickert et al. 2016; Hahn et al. 2020). The runtime to construct a GFM automaton from an LTL formula is doubly exponential in the size of the formula, which is the same cost as obtaining a deterministic automata. However, the practical advantage of going through GFM instead of deterministic automata is enormous and all state-of-the-art systems implement an LTL to GFM construction (Sickert et al. 2016; Hahn et al. 2020).

4 Probabilistic ω-Automata

Our state reduction technique is based on obtaining a small probabilistic (Büchi) automaton (PA) out of a GFM automaton. The key characteristic of PAs (Baier, Größer, and Bertrand 2012) is that, while being nondeterministic, their nondeterministic choices are resolved randomly. A PA $\mathcal{P} = (\Sigma, Q, \delta, q_0, \alpha)$ is a nondeterministic automaton equipped with a Büchi acceptance condition α and a randomised transition function $\delta : Q \times \Sigma \mapsto \mathsf{Distr}(Q)$, where, from state q and letter σ, transition to q' is taken with probability $\delta(q, \sigma)(q')$. We often abuse notation by writing $\delta(q, \sigma)$ to denote its support. Each word $w \in \Sigma^\omega$ induces a probability measure $\mathbb{P}_{\mathcal{P}}^w$ on Q^ω in the usual way. The probability that $\mathcal{P}$ accepts w, denoted by $\mathbb{P}_{\mathcal{P}}(w)$, is the probability measure of all accepting runs of w on $\mathcal{P}$, that is:

$$\mathbb{P}_{\mathcal{P}}(w) = \mathbb{P}_{\mathcal{P}}^w(\{\pi : \pi \text{ is an accepting run of } w\}).$$

A PA $\mathcal{P}$ is a 0/1-PA if, for any word $w \in \Sigma^\omega$, we have either $\mathbb{P}_{\mathcal{P}}(w) = 1$ or $\mathbb{P}_{\mathcal{P}}(w) = 0$. For a 0/1-PA $\mathcal{P}$, with a slight abuse of notation, we say a word w is accepted by $\mathcal{P}$ (w is in $\mathcal{L}(\mathcal{P})$) if $\mathbb{P}_{\mathcal{P}}(w) = 1$. 0/1-PAs are known to be *semantically deterministic* (Li et al. 2025). That is, given a state p and a letter σ of 0/1-PA $\mathcal{P}$, for any two states $q, r \in \delta(p, \sigma)$, we have that $\mathcal{L}(\mathcal{P}^q) = \mathcal{L}(\mathcal{P}^r)$ where $\mathcal{P}^s$ is the automaton by setting the initial state to s. For a detailed introduction to PAs please refer to (Baier, Größer, and Bertrand 2012).

Product of MDPs and 0/1-PA. In the following, we show for the *first* time that 0/1-PAs can also be perceived as another type of GFM automata. To this end, we first define the product MDP of an MDP $\mathcal{M}$ and a 0/1-PA $\mathcal{P}$ [2].

Formally we define the product as follows. Let $\mathcal{M} = (S, \mathsf{Act}, \Sigma, \mathsf{P}, s_0, L)$ be an MDP and $\mathcal{P} = (Q, q_0, \delta, \alpha)$ be the 0/1-PA over Σ. We define the product MDP $\mathcal{M} \otimes \mathcal{P} = (S^{\otimes}, \mathsf{Act}^{\otimes}, \Sigma, \mathsf{P}^{\otimes}, \langle s_0, q_0 \rangle, L^{\otimes}, \alpha^{\otimes})$ where

- $S^{\otimes} = S \times Q$ is the state space,
- $\mathsf{Act}^{\otimes} = \mathsf{Act}$ is the action set,
- $\mathsf{P}^{\otimes} : S^{\otimes} \times \mathsf{Act}^{\otimes} \times S^{\otimes} \to [0, 1]$ is the transition probability function such that $\mathsf{P}^{\otimes}(\langle s, q \rangle, a, \langle s', q' \rangle) = \mathsf{P}(s, a, s') \cdot \delta(q, \sigma, q')$ if $\mathsf{P}(s, a, s') > 0$ and $q' \in \delta(q, \sigma)$ where $\sigma = L(s, a)$,
- $L^{\otimes} : S^{\otimes} \times \mathsf{Act}^{\otimes} \to \Sigma$ where $L^{\otimes}(\langle s, q \rangle, a) = L(s, a)$ and
- $\alpha^{\otimes} = \{(\langle s, q \rangle, a, \langle s', q' \rangle) \mid (q, L(s, a), q') \in \alpha, (q, \sigma, q') \in \alpha, a \in \mathsf{Act}^{\otimes}, \mathsf{P}^{\otimes}(\langle s, q \rangle, a, \langle s', q' \rangle) > 0\}$.

Intuitively, we still ask the agent to make decisions on which letter to choose for the 0/1-PA $\mathcal{P}$, but leave the choice of a successor to a random strategy. That is, the agent resolves the nondeterminism by choosing actions, and once the action a is selected, according to the definition of MDP $\mathcal{M} \otimes \mathcal{P}$, we also know the chosen letter $\sigma = L^{\otimes}(\langle s, q \rangle, a)$ as well. This is different from the classical product $\mathcal{M} \times \mathcal{A}$, where the agent not only decides the next letter, but also the next successor in $\mathcal{A}$, by selecting the action $\langle a, i \rangle \in \mathsf{Act}^{\times}$.

For the product $\mathcal{M} \otimes \mathcal{P}$, we can define a similar maximal *syntactic* satisfaction probability:

$$\mathsf{Psyn}(\mathcal{M}, \mathcal{P}) = \sup_{\mu} \mathbb{P}\{\xi \in \Omega_{\mu}^{\mathcal{M} \otimes \mathcal{P}}(\langle s_0, q_0 \rangle) : \xi \text{ is accepting}\}.$$

Crucially, we can see that 0/1-PAs are another type of GFM Büchi automata in the following sense:

Theorem 2. *Let $\mathcal{D}$ be a DBA and $\mathcal{P}$ be its equivalent 0/1-PA such that $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{D})$. For a finite MDP $\mathcal{M}$, we have that $\mathsf{Psyn}(\mathcal{M}, \mathcal{P}) = \mathsf{Psyn}(\mathcal{M}, \mathcal{D}) = \mathsf{Psem}(\mathcal{M}, \mathcal{D}) = \mathsf{Psem}(\mathcal{M}, \mathcal{P})$.*

An equivalent 0/1-PA $\mathcal{P}$ always exists for a DBA $\mathcal{D}$ because a DBA can be easily transformed to an equivalent 0/1-PA by transitioning to the only successor with probability one. Note that we only consider the Büchi condition in Theorem 2, but the results can easily be generalised to Rabin conditions. Since 0/1-PAs are GFM based on our product MDP definition, we can then apply standard strategy synthesis approach once the product MDP $\mathcal{M} \otimes \mathcal{P}$ is constructed.

5 Good-for-MDP State Reduction

In this section, we present our main contribution, a general state-space reduction for GFM automata. We assume to have a GFM automaton $\mathcal{A}_{\text{GFM}}$ and through a polynomial *Redux* procedure, we return a 0/1-PA $\mathcal{A}_{\text{PA}}$, whose state space can be significantly reduced. *Redux* works in several stages, see Figure 1, which we detail below.

[2] We note that a source of 0/1-PAs can be a special type of automata whose nondeterminism can be resolved by randomness discussed in (Henzinger, Prakash, and Thejaswini 2025). This type of automata is also said to be GFM in a discussion without formal proofs in (Henzinger, Prakash, and Thejaswini 2025).

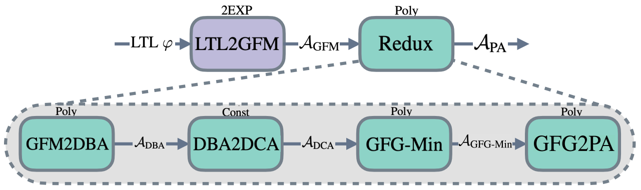
2EXP
Poly
LTL φ
LTL2GFM
$\mathcal{A}_{\text{GFM}}$
Redux
$\mathcal{A}_{\text{PA}}$
Poly
Const
Poly
Poly
GFM2DBA
$\mathcal{A}_{\text{DBA}}$
DBA2DCA
$\mathcal{A}_{\text{DCA}}$
GFG-Min
$\mathcal{A}_{\text{GFG-Min}}$
GFG2PA

Figure 1: Overview of our state reduction pipeline

- First step (Poly): we perform a GFM-to-DBA transformation by embedding in the transitions of the GFM $\mathcal{A}_{\text{GFM}}$ the choices $[k]$ used in the product construction of Section 3, making it a complete DBA $\mathcal{A}_{\text{DBA}}$.
- Second step (Const): we treat the DBA $\mathcal{A}_{\text{DBA}}$ as a DCA, $\mathcal{A}_{\text{DCA}}$, by simply changing the acceptance condition.
- Third step (Poly): we apply a minimisation algorithm originally developed for good-for-games (GFG) NCAs (Radi and Kupferman 2022) to $\mathcal{A}_{\text{DCA}}$ to obtain a minimal GFG-NCA $\mathcal{A}_{\text{GFG-Min}}$.
- Fourth step (Poly): we transform the GFG-NCA $\mathcal{A}_{\text{GFG-Min}}$ into a 0/1-PA $\mathcal{A}_{\text{PA}}$ with the same number of states as $\mathcal{A}_{\text{GFG-Min}}$.

Here "Poly" and "Const" indicate that their corresponding steps perform in polynomial and constant time, respectively. We now detail each step below.

Step 1: GFM to DBA

We consider the product $\mathcal{M} \times \mathcal{A}_{\text{GFM}}$ of Section 3 and modify both the MDP $\mathcal{M}$ and automaton $\mathcal{A}_{\text{GFM}}$ by embedding the transition choices $[k]$. The resulting MDP $\mathcal{M}'$ has its action set expanded from Act to $\mathsf{Act} \times [k]$. Similarly, the resulting automaton $\mathcal{A}_{\text{DBA}}$ will have the transitions re-labelled from Σ to $\Sigma \times [k]$. As a result the automaton $\mathcal{A}_{\text{DBA}}$ is deterministic.

Formally, we define the modified MDP $\mathcal{M}' = (S', s'_0, \mathsf{Act}', \mathsf{P}', L')$ from $\mathcal{M} = (S, s_0, \mathsf{Act}, \mathsf{P}, L)$ as

- $S' = S$, $s'_0 = s_0$, $\mathsf{Act}' = \mathsf{Act} \times [k]$,
- $L' : S' \times \mathsf{Act}' \to \Sigma'$ where $\Sigma' = \Sigma \times [k]$, and if $L(s, a) = \sigma$, then $L'(s, \langle a, i \rangle) = \langle \sigma, i \rangle$ for all $i \in [k]$, and
- $\mathsf{P}'(s, \langle a, i \rangle, s') = \mathsf{P}(s, a, s')$ for all $i \in [k]$ and $\mathsf{P}(s, a, s') > 0$.

We define the DBA $\mathcal{A}_{\text{DBA}} = (Q_{\text{DBA}}, q_0, \delta_{\text{DBA}}, \alpha_{\text{DBA}})$ from $\mathcal{A}_{\text{GFM}} = (Q, q_0, \delta, \alpha)$ as:

- $Q_{\text{DBA}} = Q$, $\alpha_{\text{DBA}} = \{(q, \langle \sigma, i \rangle, q') \mid (q, \sigma, q') \in \alpha, i = \mathsf{idx}_{\mathcal{A}, q, \sigma}(q')\}$, and
- $\delta_{\text{DBA}}(q, \langle \sigma, i \rangle) = q'$ if $q' \in \delta(q, \sigma)$ and $\mathsf{idx}_{\mathcal{A}, q, \sigma}(q') = i$.

$\mathcal{A}_{\text{DBA}}$ will then be made complete. Intuitively, we assign the nondeterminism of $\mathcal{A}_{\text{GFM}}$ to the agent by extending the action set Act to $\mathsf{Act} \times [k]$ in $\mathcal{M}'$, which then would allow us to obtain the DBA $\mathcal{A}_{\text{DBA}}$ since the nondeterminism in $\mathcal{A}_{\text{GFM}}$ will be taken care of by the agent. We observe that $\mathcal{M} \times \mathcal{A}_{\text{GFM}}$ and $\mathcal{M}' \times \mathcal{A}_{\text{DBA}}$ (obtained by applying the product construction in Section 3) have the same state space. Moreover, in $\mathcal{M}' \times \mathcal{A}_{\text{DBA}}$, the action set is $\mathsf{Act} \times [k] \times [1]$, as the maximal out-degree of the DBA $\mathcal{A}_{\text{DBA}}$ is 1. Notice that we can drop $[1]$, which is a constant in every transition and if we do so, we get the following:

Lemma 1. *The product $\mathcal{M} \times \mathcal{A}_{GFM}$ and $\mathcal{M}' \times \mathcal{A}_{DBA}$ are the same. Moreover, the optimal strategy μ for each state pair $(s, q) \in S \times Q$ in either of the products will obtain optimal satisfaction probabilities in both products.*

By Lemma 1, it is easy to observe that the syntactic probabilities of $\mathcal{M} \times \mathcal{A}_{\mathrm{GFM}}$ and $\mathcal{M}' \times \mathcal{A}_{\mathrm{DBA}}$ are the same. Since $\mathcal{A}_{\mathrm{DBA}}$ is deterministic and thus GFM, we have $\mathsf{Psyn}(\mathcal{M}', \mathcal{A}_{\mathrm{DBA}}) = \mathsf{Psem}(\mathcal{M}', \mathcal{A}_{\mathrm{DBA}})$. It then follows:

Theorem 3. $\mathsf{Psem}(\mathcal{M}, \mathcal{A}_{GFM}) = \mathsf{Psyn}(\mathcal{M}, \mathcal{A}_{GFM}) = \mathsf{Psyn}(\mathcal{M}', \mathcal{A}_{DBA}) = \mathsf{Psem}(\mathcal{M}', \mathcal{A}_{DBA})$.

The GFM to DBA conversion can be done in polynomial time. The construction of the modified MDP $\mathcal{M}'$ and the DBA $\mathcal{A}_{\mathrm{DBA}}$ involves a direct translation of the original components. The number of states remains the same, while the size of the action set and the alphabet expands by a factor of k, which is a polynomial increase.

Step 2: DBA to DCA

Next, we syntactically convert the DBA $\mathcal{A}_{\mathrm{DBA}}$ into a DCA $\mathcal{A}_{\mathrm{DCA}}$. To do so, we only need to modify the acceptance condition. $\mathcal{A}_{\mathrm{DCA}}$ has the same set of states, initial state, and transition function as $\mathcal{A}_{\mathrm{DBA}}$. The co-Büchi acceptance condition requires that a run is accepting if it intersects with the set of rejecting transitions only a finite number of times. The set α_{DCA} of rejecting transitions for the DCA $\mathcal{A}_{\mathrm{DCA}}$ is the same as the set α_{DBA} of accepting transitions in $\mathcal{A}_{\mathrm{DBA}}$, but only interpreted differently. Observe that the languages of $\mathcal{A}_{\mathrm{DBA}}$ and $\mathcal{A}_{\mathrm{DCA}}$ complement each other. $\mathcal{A}_{\mathrm{DCA}}$ is just an intermediate automaton on which we apply GFG-minimisation next.

Since this conversion does not require any computational modification to the automaton's transition structure, it can be done in constant runtime.

Step 3: GFG-Minimisation

$\mathcal{A}_{\mathrm{DCA}}$ is deterministic and thus a GFG automaton that we can apply minimisation algorithms (Radi and Kupferman 2022) on in this step. To formally understand this step, we take a detour to introduce GFG automata.

GFG automata (Henzinger and Piterman 2006) are automata in which nondeterminism can be resolved *deterministically* by assigning the choice to the protagonist in an adversarial (vs. stochastic MDPs) games. Formally, an automaton $\mathcal{A}$ is said to be GFG if there exists a strategy $f : \Sigma^* \to Q$ such that for every accepting word $w = \sigma_0\sigma_1\cdots$, the run $f(\epsilon)f(\sigma_0)\cdots f(\sigma_0\cdots\sigma_i)\cdots$ is accepting. Intuitively, this indicates that there is a deterministic strategy to produce an accepting run in a GFG automaton even if the input accepting word is given letter by letter. Interestingly, GFG co-Büchi automata can be minimised in *polynomial* time (Radi and Kupferman 2022). Since GFG automata have more restricted nondeterminism than GFM automata, GFG automata are also GFM (Klein et al. 2014). GFG automata are not adopted in state-of-the-art stochastic planning tools, such as PRISM (Kwiatkowska, Norman, and Parker 2011). The reason is that GFG Büchi (resp. co-Büchi) automata cannot recognise all languages expressed by LTL as they are only as expressive as their deterministic counterparts. Moreover, current approaches to construct GFG automata are not as efficient as the ones for GFM. Notably, our work is able to use GFG minimisation on GFM automata that recognise all ω-regular properties. See Appendix G for more details.

In our construction we apply GFG co-Büchi minimisation (Radi and Kupferman 2022) on $\mathcal{A}_{\mathrm{DCA}}$, yielding a minimal GFG-NCA $\mathcal{A}_{\text{GFG-Min}}$ accepting $\mathcal{L}(\mathcal{A}_{\mathrm{DCA}})$.

Step 4: GFG-Min to 0/1-PA

(Li et al. 2025) has proven that the minimal GFG-NCA produced by (Radi and Kupferman 2022) can be turned into a language equivalent 0/1-PA by resolving its nondeterminism with random choices.

The translation from the minimal GFG-NCA $\mathcal{A}_{\text{GFG-Min}}$ to a 0/1-PA $\mathcal{A}_{\mathrm{PA}}$ is quite simple: we only need to resolve the nondeterminism by random choices. Formally, we treat $\mathcal{A}_{\text{GFG-Min}}$ as a Büchi automaton $\mathcal{A}_{\mathrm{NBA}} = (Q_{\mathrm{NBA}}, q_0, \delta_{\mathrm{NBA}}, \alpha_{\mathrm{NBA}})$ and build the PA $\mathcal{A}_{\mathrm{PA}} = (Q_{\mathrm{PA}}, q_0, \delta_{\mathrm{PA}}, \alpha_{\mathrm{PA}})$ as:

- $Q_{\mathrm{PA}} = Q_{\mathrm{NBA}}, \alpha_{\mathrm{PA}} = \alpha_{\mathrm{NBA}}$, and
- $\delta_{\mathrm{PA}}(q, \sigma)(q') = \frac{1}{|\delta_{\mathrm{NBA}}(q,\sigma)|}$ for each $q \in Q_{\mathrm{PA}}, \sigma \in \Sigma'$ and $q' \in \delta_{\mathrm{NBA}}(q, \sigma)$.

Following from (Li et al. 2025, Lemma 3), we get:

Lemma 2. $\mathcal{L}(\mathcal{A}_{PA}) = \mathcal{L}(\mathcal{A}_{DBA})$ *and* $\mathcal{A}_{PA}$ *is a 0/1-PA.*

It immediately follows that, according to Theorem 2, we can use $\mathcal{A}_{\mathrm{PA}}$ to obtain an optimal strategy for $\mathcal{M}'$ to achieve $\mathsf{Psem}(\mathcal{M}', \mathcal{A}_{\mathrm{PA}}) = \mathsf{Psem}(\mathcal{M}', \mathcal{A}_{\mathrm{DBA}})$ since $\mathcal{L}(\mathcal{A}_{\mathrm{PA}}) = \mathcal{L}(\mathcal{A}_{\mathrm{DBA}})$ holds. Together with Theorems 2 and 3, we then obtain our main result:

Theorem 4. *From an optimal strategy μ on $\mathcal{M}' \otimes \mathcal{A}_{PA}$, one can obtain an optimal strategy μ' for $\mathcal{M}$ that achieves the maximal probability $\mathsf{Psem}(\mathcal{M}, \mathcal{A}_{GFM})$.*

Summary. Details on optimal strategy synthesis, along with further optimisations, are in Appendix L. In summary, our stochastic planning algorithm can be formalised below:

- Construct a GFM (Büchi) automaton $\mathcal{A}_{\mathrm{GFM}}$ from φ.
- Create $\mathcal{M}'$ from $\mathcal{M}$ and $\mathcal{A}_{\mathrm{DBA}}$ (and thus $\mathcal{A}_{\mathrm{DCA}}$) from $\mathcal{A}_{\mathrm{GFM}}$.
- Apply GFG minimisation on $\mathcal{A}_{\mathrm{DCA}}$ and treat the minimised automaton $\mathcal{A}_{\text{GFG-Min}}$ as a Büchi automaton $\mathcal{A}_{\mathrm{NBA}}$.
- Translate $\mathcal{A}_{\mathrm{NBA}}$ to a 0/1-PA $\mathcal{A}_{\mathrm{PA}}$.
- Create $\mathcal{M}' \otimes \mathcal{A}_{\mathrm{PA}}$.
- Identify the accepting MECs that have some accepting transitions.
- Synthesise an optimal strategy μ that maximises the reachability probability of accepting MECs as usual.

6 Direct GFM Construction for $\mathsf{GF}\varphi$

$\mathsf{GF}\varphi$ fragment. We now focus on the syntactic class of LTL formulas of the form $\mathsf{GF}\varphi$ where φ is a *co-safety* formula with the following syntax:

$$\varphi := a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \mathsf{F}\varphi \mid \varphi \mathsf{U} \varphi.$$

They express that a finite trace pattern repeats infinitely often, which we refer to as *repeated reachability* properties.

Formally, $P \subseteq \Sigma^\omega$ is a repeated reachability property if there exists a language of finite words $R \subseteq \Sigma^*$ such that, for every $w \in P$, infinitely many prefixes of w belong to $\Sigma^* \cdot R$. Then, we have that:

Lemma 3. *P is a repeated reachability property specifiable in LTL if, and only if, P is specifiable in $\mathsf{GF}\varphi$, where φ is a co-safety formula.*

The $\mathsf{GF}\varphi$ fragment is very common in verification (Holeček et al. 2004) and also in reinforcement learning (Jackermeier and Abate 2025). More generally, repeated reachability properties are within a commonly used subclass of so-called *recurrence properties* (Manna and Pnueli 1990; Chang, Manna, and Pnueli 1992).

Direct GFM Construction for $\mathsf{GF}\varphi$. Consider a formula of the form $\mathsf{GF}\varphi$, the subformula φ expresses a *finite trace property*, which can be accepted by a nondeterministic finite automaton (NFA)[3]. To make our construction more general, we will start from an NFA instead of the formula φ in the following. An NFA $\mathcal{N}$ is a tuple $(\mathcal{T}, F)$ where $\mathcal{T} = (Q, q_0, \delta)$ is a *nondeterministic* TS and $F \subseteq Q$ is a set of *final* states. Unlike NBAs, NFAs only operate on *finite* traces rather than ω-traces. A finite trace $u \in \Sigma^*$ is accepted by an NFA $\mathcal{N}$ if one of its finite runs *terminates* at a final state.

We can construct for a co-safety property formula φ, an NFA $\mathcal{N}_\varphi = (Q, q_0, \delta, F)$ such that $\mathcal{L}_*(\mathcal{N}_\varphi) \cdot (tt)^\omega = [\varphi]$ where $\mathcal{L}_*(\mathcal{N})$ denotes the set of finite traces accepted by $\mathcal{N}_\varphi$. Note that the alphabet here is $\Sigma = 2^{\mathsf{AP}}$.

Now we can construct from $\mathcal{N}_\varphi$ a GFM automaton $\mathcal{A} = (Q_\mathcal{A}, q_\mathcal{A}, \delta_\mathcal{A}, \alpha_\mathcal{A})$ for $\mathsf{GF}\varphi$ where

- $Q_\mathcal{A} \subseteq ((Q \setminus F) \cup \{q_0\})$, $q_\mathcal{A} = q_0$,
- $\delta_\mathcal{A} : Q_\mathcal{A} \times \Sigma \to 2^{Q_\mathcal{A}}$ is defined such that for each $q \in Q_\mathcal{A}$ and $\sigma \in \Sigma$, we have (1) $q' \in \delta_\mathcal{A}(q, \sigma)$ if $q' \in \delta(q, \sigma)$ with $q' \notin F$, and (2) $q_0 \in \delta_\mathcal{A}(q, \sigma)$, and
- $\alpha_\mathcal{A} = \{(q, \sigma, q_0) \in \Delta(\delta_\mathcal{A}) \mid \exists q' \in \delta(q, \sigma) \wedge q' \in F\}$.

The intuition is that, for a repeated reachability property $\mathsf{GF}\varphi$, we can forget the past finite trace at *any* point and start tracking whether the following finite trace satisfies φ. The Büchi acceptance condition will make sure that the reachability/co-safety formula φ will be satisfied infinitely often. Therefore, in the construction, we can always reset and go back to the initial state q_0, which is why every state q has a successor q_0 on every letter $\sigma \in \Sigma$. Once the formula φ has been fulfilled, i.e., a final state q' has been reached from q over letter σ in $\mathcal{N}_\varphi$, we need to mark the transition (q, σ, q_0) in $\mathcal{A}$ as accepting and start tracking finite traces satisfying φ again by moving back to the initial state q_0.

It is also easy to show that a memoryless random strategy on $\mathcal{A}$ can generate an accepting run almost surely over an ω-trace from $[\mathsf{GF}\varphi]$. This is because accepting transitions can be reached with positive probabilities from anywhere in $\mathcal{A}$ and it is not possible to skip accepting transitions forever in an infinite run with positive probabilities.

[3]Our construction applies to any NFA, not just those from co-safety LTL (see Appendix K), covering repeated reachability properties not expressible in LTL, i.e., monadic first-order and finite LTL (De Giacomo and Vardi 2013).

Pattern	`Owl`	`Owl-Red`	`Slim`	`Slim-Red`
TDR[6]	64 (0.46)	64 (0.03)	65 (0.09)	**34** (0.01)
TDR[7]	128 (0.46)	128 (0.15)	129 (0.12)	**66** (0.02)
TDR[8]	256 (0.48)	256 (0.69)	257 (0.14)	**130** (0.05)
LIB[4]	17 (0.44)	**10** (0.01)	33 (0.11)	18 (0.13)
LIB[5]	21 (0.48)	**12** (0.01)	65 (0.20)	34 (3.83)
LIB[6]	25 (0.62)	**14** (0.01)	129 (0.69)	66 (128.34)
BRP[6]	69 (0.55)	**17** (0.02)	317 (0.14)	65 (0.17)
BRP[7]	133 (0.65)	**19** (0.04)	637 (0.21)	95 (0.71)
BRP[8]	261 (0.86)	**21** (0.14)	1277 (0.33)	146 (3.49)
EHP	13 (0.43)	**9** (0.01)	127 (0.14)	54 (0.31)
NU[4]	44 (1.16)	**23** (0.02)	51 (0.22)	39 (0.02)
NU[5]	150 (1.41)	**56** (0.17)	232 (0.82)	159 (0.76)
NU[6]	433 (5.92)	**249** (27.75)	1425 (12.24)	753 (163.32)
LFR[6]	39 (0.77)	40 (0.27)	40 (0.44)	**21** (0.19)
LFR[7]	72 (0.92)	73 (2.99)	73 (1.57)	**34** (1.76)
LFR[8]	137 (1.45)	138 (39.11)	138 (10.74)	**59** (20.84)

Table 1: Comparison of automata state spaces of `Owl` and `Slim` against their reduced versions, `Owl-Red` and `Slim-Red`. We report number of states (smallest in bold) and runtime (in seconds). Within `Owl-Red` and `Slim-Red` we only measure the time it took to reduce the automaton.

Let $|\varphi|$ be the number of modalities and connectives in φ. It then follows that:

Theorem 5. *(1) $\mathcal{L}(\mathcal{A}) = [\mathsf{GF}\varphi]$, (2) $\mathcal{A}$ is GFM and (3) $\mathcal{A}$ has $2^{\mathcal{O}(|\varphi|)}$ states.*

Note that as, for a co-safety/reachability property φ, the NFA $\mathcal{N}_\varphi$ of φ has $2^{\mathcal{O}(|\varphi|)}$ states (De Giacomo and Vardi 2013), so does our GFM automaton $\mathcal{A}$ for $\mathsf{GF}\varphi$. Current GFM constructions such as (Sickert et al. 2016; Hahn et al. 2020) normally output a DBA for $\mathsf{GF}\varphi$, which in general has $2^{2^{\mathcal{O}(|\varphi|)}}$ states. Therefore, our specialised GFM construction for repeated reachability property $\mathsf{GF}\varphi$ can be *exponentially more efficient* than current approaches.

7 Experiments for GFM state reduction

In this section, we validate the performance of our GFM state-space reduction on LTL specifications extracted from literature, which we define blow. The full benchmark set, demonstrating consistent state-space reduction on complex LTL specifications from over 8 sources, is in Appendix N.

Trigger with Delayed Response (TDR). This pattern requires a followed by b after n steps, to hold infinitely often. We define this fragment as $\text{TDR}[n] = \mathsf{GF}(a \wedge \mathsf{X}^n b)$ where X^n denotes n nested applications of the next operator.

Liberouter (LIB). An example from the Liberouter verification project (Holeček et al. 2004) checks liveness of binary signals, ensuring at least one signal does not become permanently stuck. Formally, $\text{LIB}[n] = \mathsf{GF}((a \wedge \mathsf{X}\neg a) \vee (\neg a \wedge \mathsf{X}a) \vee (b \wedge \mathsf{X}\neg b) \vee (\neg b \wedge \mathsf{X}b) \dots)$, with n numbers of signals.

Bounded Retransmission Protocol (BRP). Based on (Baier et al. 2023), BRP models that whenever a message is sent, a corresponding acknowledgement must occur within a bounded number of steps. BRP$[n] = \mathsf{G}(\texttt{"msg_sent"} \rightarrow \mathsf{F}(\texttt{"ack_send"} \wedge \varphi_n))$, where the subformula φ_n ensures that $\texttt{"ack\_rev"}$ occurs within n steps after $\texttt{"ack\_send"}$. We define $\varphi_n = \texttt{"ack\_rev"} \vee \mathsf{X}(\texttt{"ack\_rev"} \vee \mathsf{X}(\cdots \vee \mathsf{X}(\texttt{"ack\_rev"})))$ with X applied n times. A smaller n enforces stricter guarantees, while a larger n relaxes them.

Etessami–Holzmann Patterns (EHP). To evaluate the effectiveness of Büchi automata optimisations, (Etessami and Holzmann 2000) present deeply nested specifications. One formula is given by EHP $= a\mathsf{U}(b \wedge \mathsf{X}(c \wedge \mathsf{F}(d \wedge \mathsf{XF}(e \wedge \mathsf{XF}(f \wedge \mathsf{XF}g)))))$. It expresses that condition a must hold continuously until a specific sequence of events is triggered.

Nested Until Dependencies (NU). We define the nested-until pattern NU$[n] = \mathsf{G}(p_1 \rightarrow \varphi_n)$ where φ_k is recursively defined by $\varphi_k = p_k \mathsf{U} \varphi_{k+1}$ for $k < n$, and $\varphi_n = p_n \mathsf{U} p_{n+1}$. This pattern was used in the runtime verification benchmarks for SystemC models (Tabakov, Rozier, and Vardi 2012).

Layered Fairness and Reachability (LFR). As part of their evaluation, (Müller and Sickert 2017) introduced LFR$[n] = \mathsf{F}(b_1) \wedge (\mathsf{F}(b_2 \wedge \ldots \mathsf{F}(b_n)) \wedge \mathsf{GF}(a_1 \wedge \mathsf{X}(a_2 \wedge \ldots \mathsf{X}(a_n)))$. The left-hand conjunct captures a nested reachability, while the right-hand side encodes a fairness condition, requiring $a_1, \ldots, a_n$ to reoccur, with strict temporal progression.

Experiment Setup. In order to translate these LTL specifications to GFM automata, we use two state-of-the-art approaches. Owl (Kretínský, Meggendorfer, and Sickert 2018), which we denote by `Owl`, and the "slim" GFM construction of (Hahn et al. 2020), which we implemented ourselves and denote by `Slim`. We refer to reduced Owl automata as `Owl-Red` and to reduced "slim" automata as `Slim-Red`. We use SPOT (Duret-Lutz 2013) for parts of patterns, for the GFG minimisation and we have used LTL datasets available in the Owl repository. All experiments were run on an 8-core ARM chip, with 16GB of RAM.

Results. Table 1 compares automata states before and after our reduction. We report the LTL to GFM construction time (in seconds) for `Owl` and `Slim` , and reduction time (excluding initial construction) for `Owl-Red` and `Slim-Red`.

Across all LTL patterns, the smallest automata (marked in bold) are consistently reduced ones. `Owl` features advanced formula simplification and post-processing optimisations and therefore constructs smaller automata than the "slim" GFM construction. Since our "slim" GFM construction is not as optimised as Owl, we can consistently reduce the state space of all "slim" GFMs and for cases like TDR and LFR, we obtain smaller `Slim-Red` than `Owl-Red`.

Note that for LFR, Owl returns incomplete automata, while our reduction returns complete ones with an explicit sink. See Appendix M for details. For the remaining cases NU, EHP, BRP and LIB, the smallest automata are our `Owl-Red`. More results, can be found in Appendix N.

8 Experiments for $\mathsf{GF}\varphi$

The first two LTL specification patterns above, namely **TDR** and **LIB**, refer to formulas that have the syntactic $\mathsf{GF}\varphi$ structure and therefore, we can use them to evaluate our direct GFM automata construction. In particular, we show below that the direct construction gives us a large improvement with respect to the standard state-of-the-art GFM constructions (Kretínský, Meggendorfer, and Sickert 2018; Hahn et al. 2020), even after applying our GFM reduction.

Pattern	`Owl-Red`	`Slim-Red`	`GFM-GF`
TDR[6]	64 (0.49)	34 (0.10)	**7** (0.08)
TDR[7]	128 (0.61)	66 (0.14)	**8** (0.08)
TDR[8]	256 (1.17)	130 (0.19)	**9** (0.08)
TDR[9]	512 (4.27)	258 (0.48)	**10** (0.10)
TDR[10]	1024 (23.67)	514 (0.91)	**11** (0.09)
LIB[6]	14 (0.63)	66 (129.03)	**13** (0.09)
LIB[7]	16 (1.47)	timeout	**15** (0.12)
LIB[8]	18 (5.95)	timeout	**17** (0.20)
LIB[9]	20 (37.78)	timeout	**19** (0.23)

Table 2: State and runtime (in seconds) comparison of our direct construction `GFM-GF` against our reduced automata, `Owl-Red` and `GFM-Red`. The runtime for `Owl-Red` and `GFM-Red` includes construction and consecutive application of our reduction (in seconds). Timeout is 300 seconds.

Experiment Setup. Table 2 compares our direct GFM construction (denoted by `GFM-GF`) to the automata obtained by our GFM state-space reduction. Again, we denote number of states of reduced Owl and "slim" GFM automata (`Owl-Red`, `Slim-Red`) but in contrast to Table 1, the runtime (in seconds), now contains both LTL to GFM construction and subsequent application of our reduction.

Results. For the LTL pattern TDR, Table 2 clearly shows that our direct construction `GFM-GF`, significantly outperforms both `Owl-Red` and `Slim-Red`, both in state size and runtime. Even when comparing the state size and construction time of `GFM-GF` to `Owl` and `Slim`, we can build exponentially more succinct automata, in a fraction of the time. For the LIB pattern, `GFM-GF` builds automata with a similar amount of states than `Owl-Red`, but again, we only require a fraction of the time. LIB[7] and LIB[8] timed out during reduction, while LIB[9] timed out during automata construction. More results can be found in Appendix N.

9 Conclusion

We conclude the paper by observing that our results immediately have an impact on a wide range of applications that use GFM automata, such as obtaining smaller strategies for planning problems, improving the efficiency of probabilistic verification (Hahn et al. 2015; Sickert and Kretínský 2016), and reinforcement learning (Hahn et al. 2020; Jackermeier and Abate 2025). Furthermore, our results can be used to reduce the GFM automata obtained from variants of LTL, including LTLf+ and PPLTL+ recently proposed in (Aminof et al. 2025; De Giacomo et al. 2025).

Acknowledgements

We would like to thank the anonymous reviewers for their constructive feedback, that helped improve the paper. This work was supported in part by the CAS Project for Young Scientists in Basic Research (Grant No. YSBR-040), ISCAS Basic Research (Grant Nos. ISCAS-JCZD-202406, ISCAS-JCZD-202302), ISCAS New Cultivation Project ISCAS-PYFX-202201, the UKRI Erlangen AI Hub on Mathematical and Computational Foundations of AI (No. EP/Y028872/1) and by the EPSRC through grants EP/X03688X/1 and EP/X042596/1.

References

Aminof, B.; De Giacomo, G.; Rubin, S.; and Vardi, M. Y. 2025. LTLf+ and PPLTL+: Extending LTLf and PPLTL to Infinite Traces. In *IJCAI*.

Bacchus, F.; Boutilier, C.; and Grove, A. J. 1996. Rewarding Behaviors. In *AAAI/IAAI, Vol. 2*, 1160–1167. AAAI Press / The MIT Press.

Bacchus, F.; Boutilier, C.; and Grove, A. J. 1997. Structured Solution Methods for Non-Markovian Decision Processes. In *AAAI/IAAI*, 112–117. AAAI Press / The MIT Press.

Baier, C.; Größer, M.; and Bertrand, N. 2012. Probabilistic ω-automata. *J. ACM*, 59(1): 1:1–1:52.

Baier, C.; and Katoen, J. 2008. *Principles of model checking*. MIT Press.

Baier, C.; Kiefer, S.; Klein, J.; Müller, D.; and Worrell, J. 2023. Markov chains and unambiguous automata. *J. Comput. Syst. Sci.*, 136: 113–134.

Brafman, R. I.; and Giacomo, G. D. 2024. Regular decision processes. *Artif. Intell.*, 331: 104113.

Chang, E. Y.; Manna, Z.; and Pnueli, A. 1992. Characterization of Temporal Property Classes. In *ICALP*, volume 623 of *Lecture Notes in Computer Science*, 474–486. Springer.

de Alfaro, L. 1997. *Formal verification of probabilistic systems*. Ph.D. thesis, Stanford University, USA.

De Giacomo, G.; Li, Y.; Schewe, S.; Weinhuber, C.; and Yu, P. 2025. Solving MDPs with LTLf+ and PPLTL+ Temporal Objectives. In *IJCAI*.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 854–860. IJCAI/AAAI.

Duret-Lutz, A. 2013. Manipulating LTL Formulas Using Spot 1.0. In *ATVA*, volume 8172 of *Lecture Notes in Computer Science*, 442–445. Springer.

Dwyer, M. B.; Avrunin, G. S.; and Corbett, J. C. 1998. Property specification patterns for finite-state verification. In *FMSP*, 7–15. ACM.

Esparza, J.; and Kretínský, J. 2014. From LTL to Deterministic Automata: A Safraless Compositional Approach. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, 192–208. Springer.

Esparza, J.; Kretínský, J.; and Sickert, S. 2020. A Unified Translation of Linear Temporal Logic to ω-Automata. *J. ACM*, 67(6): 33:1–33:61.

Etessami, K.; and Holzmann, G. J. 2000. Optimizing Büchi Automata. In *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, 153–167. Springer.

Geldenhuys, J.; and Hansen, H. 2006. Larger Automata and Less Work for LTL Model Checking. In *SPIN*, volume 3925 of *Lecture Notes in Computer Science*, 53–70. Springer.

Hahn, E. M.; Li, G.; Schewe, S.; Turrini, A.; and Zhang, L. 2015. Lazy Probabilistic Model Checking without Determinisation. In *CONCUR*, volume 42 of *LIPIcs*, 354–367. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Hahn, E. M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; and Wojtczak, D. 2020. Good-for-MDPs Automata for Probabilistic Analysis and Reinforcement Learning. In *TACAS (1)*, volume 12078 of *Lecture Notes in Computer Science*, 306–323. Springer.

Henzinger, T. A.; and Piterman, N. 2006. Solving Games Without Determinization. In *CSL*, volume 4207 of *Lecture Notes in Computer Science*, 395–410. Springer.

Henzinger, T. A.; Prakash, A.; and Thejaswini, K. S. 2025. Resolving Nondeterminism with Randomness. In Gawrychowski, P.; Mazowiecki, F.; and Skrzypczak, M., eds., *50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025, August 25-29, 2025, Warsaw, Poland*, volume 345 of *LIPIcs*, 57:1–57:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Holeček, J.; Kratochvíla, T.; Řehák, V.; Šafránek, D.; Šimeček, P.; et al. 2004. Verification results in Liberouter project.

Jackermeier, M.; and Abate, A. 2025. DeepLTL: Learning to Efficiently Satisfy Complex LTL Specifications for Multi-Task RL. In *International Conference on Learning Representations (ICLR)*.

Klein, J.; Müller, D.; Baier, C.; and Klüppelholz, S. 2014. Are Good-for-Games Automata Good for Probabilistic Model Checking? In *LATA*, volume 8370 of *Lecture Notes in Computer Science*, 453–465. Springer.

Kretínský, J.; Meggendorfer, T.; and Sickert, S. 2018. Owl: A Library for ω-Words, Automata, and LTL. In *ATVA*, volume 11138 of *Lecture Notes in Computer Science*, 543–550. Springer.

Kwiatkowska, M. Z.; Norman, G.; and Parker, D. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, 585–591. Springer.

Lacerda, B.; Parker, D.; and Hawes, N. 2014. Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In *IROS*, 1511–1516. IEEE.

Li, Y.; Paul, S.; Schewe, S.; and Tang, Q. 2025. Accelerating Markov Chain Model Checking: Good-for-Games Meets Unambiguous Automata. In Piskac, R.; and Rakamarić, Z., eds., *CAV*, volume 15932 of *Lecture Notes in Computer Science*, 276–298. Springer.

Littman, M. L. 1997. Probabilistic Propositional Planning: Representations and Complexity. In *AAAI/IAAI*, 748–754. AAAI Press / The MIT Press.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The Computational Complexity of Probabilistic Planning. *J. Artif. Intell. Res.*, 9: 1–36.

Manna, Z.; and Pnueli, A. 1990. A Hierarchy of Temporal Properties. In *PODC*, 377–410. ACM.

Meyer, P. J.; Sickert, S.; and Luttenberger, M. 2018. Strix: Explicit Reactive Synthesis Strikes Back! In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, 578–586. Springer.

Müller, D.; and Sickert, S. 2017. LTL to Deterministic Emerson-Lei Automata. In *GandALF*, volume 256 of *EPTCS*, 180–194.

Pnueli, A. 1977. The Temporal Logic of Programs. In *FOCS*, 46–57. IEEE Computer Society.

Radi, B. A.; and Kupferman, O. 2022. Minimization and Canonization of GFG Transition-Based Automata. *Log. Methods Comput. Sci.*, 18(3).

Safra, S. 1988. On the Complexity of omega-Automata. In *FOCS*, 319–327. IEEE Computer Society.

Schewe, S.; Tang, Q.; and Zhanabekova, T. 2023. Deciding What Is Good-for-MDPs. In *CONCUR*, volume 279 of *LIPIcs*, 35:1–35:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Schillinger, P.; Bürger, M.; and Dimarogonas, D. V. 2019. Hierarchical LTL-task mdps for multi-agent coordination through auctioning and learning. *The international journal of robotics research*.

Sickert, S.; and Esparza, J. 2020. An Efficient Normalisation Procedure for Linear Temporal Logic and Very Weak Alternating Automata. In *LICS*, 831–844. ACM.

Sickert, S.; Esparza, J.; Jaax, S.; and Kretínský, J. 2016. Limit-Deterministic Büchi Automata for Linear Temporal Logic. In *CAV (2)*, volume 9780 of *Lecture Notes in Computer Science*, 312–332. Springer.

Sickert, S.; and Kretínský, J. 2016. MoChiBA: Probabilistic LTL Model Checking Using Limit-Deterministic Büchi Automata. In *ATVA*, volume 9938 of *Lecture Notes in Computer Science*, 130–137.

Tabakov, D.; Rozier, K. Y.; and Vardi, M. Y. 2012. Optimized temporal monitors for SystemC. *Formal Methods Syst. Des.*, 41(3): 236–268.

Thiébaux, S.; Gretton, C.; Slaney, J. K.; Price, D.; and Kabanza, F. 2006. Decision-Theoretic Planning with non-Markovian Rewards. *J. Artif. Intell. Res.*, 25: 17–74.

Vardi, M. Y.; and Wolper, P. 1986. An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report). In *LICS*, 332–344. IEEE Computer Society.

Yang, C.; Littman, M. L.; and Carbin, M. 2022. On the (In)Tractability of Reinforcement Learning for LTL Objectives. In *IJCAI*, 3650–3658. ijcai.org.

Yu, P.; Li, Y.; Parker, D.; and Kwiatkowska, M. 2025. Planning with Linear Temporal Logic Specifications: Handling Quantifiable and Unquantifiable Uncertainty. *CoRR*, abs/2502.19603.

A Related Work

Note that our transformation procedure from a GFM automaton $\mathcal{A}_{\mathrm{GFM}}$ to a 0/1-PA $\mathcal{A}_{\mathrm{PA}}$ is similar to the one that converts a unambiguous Büchi automaton (UBA) to a 0/1-PA proposed in (Li et al. 2025). Nonetheless, (Li et al. 2025) considers only the UBAs as input while we consider GFM automata. Further, (Li et al. 2025) works only on Markov chains while we consider MDPs, which is a more general model and thus makes our algorithm also work on MCs in probabilistic verification. Hence, our work can be seen as a generalisation to (Li et al. 2025) in this regard.

According to (Schewe, Tang, and Zhanabekova 2023, Theorem 10), minimising GFM automata is PSPACE-hard. Instead of proposing GFM minimisation algorithms, we propose to use polynomial GFG minimisation to reduce the input GFM automata to 0/1-PAs, which although do not accept the original languages but can be used for analysing MDPs with our product MDP definition.

(Klein et al. 2014) showed that GFG automata can be used for analysing MDPs but current constructions give GFG automata even larger than their deterministic counterparts. This hinders the use of GFG automata for probabilistic verification in practice. Our work is the first successful effort to combine the strengths of GFG automata and GFM automata. That is, we use the polynomial GFG minimisation algorithm (Radi and Kupferman 2022) to obtain automata that are good for MDPs. Our algorithm not only gives smaller strategies for stochastic MDP planing problems but also have the potential to be more efficient in particular when the input MDPs are large.

B Action Indexing

We note that the action set Act $\times Q$ is often used for $\mathcal{M} \times \mathcal{A}$ in other literature, e.g. (Schewe, Tang, and Zhanabekova 2023). That is, the traditional product $\mathcal{M} \times \mathcal{A}$ resolves the nondeterminism in $\mathcal{A}$ by asking the agent to make the choice of the successors q in the action $\langle a, q\rangle$. Our observation is that, instead of explicitly storing the *selected* successor in the action names, we only need to know the relative position i of the chosen successor in the predefined order of $\delta(q,\sigma)$, where $q \in Q$ and $\sigma \in \Sigma$. To uniquely associate a successor q' with its relative position i in $\delta(q,\sigma)$, we need an order over the successors. A simple way to define an order is to use a *global* order over Q, such as ordering $Q = \{q_0, \cdots, q_n\}$ by their index names, but we can also define a different *local* order for each pair of q and σ. We assume that for a state q and a letter σ in $\mathcal{A}$, we have a function $\mathsf{idx}_{\mathcal{A},q,\sigma} : \delta(q,\sigma) \to \mathbb{N}$ that maps a state q' in $\delta(q,\sigma)$ of $\mathcal{A}$ to its unique position $i \in [|\delta(q,\sigma)|]$, i.e. $\mathsf{idx}_{\mathcal{A},q,\sigma}(q') = i$. In this way, our product definition is in fact equivalent to the one using the action set Act $\times Q$, but has a slight advantage of using fewer actions.

C Non-GFM Example

Figure 2 and Figure 3 display two nondeterministic Büchi automata. The NBA in Figure 2 is not GFM, while the NBA in Figure 3 is GFM. The NBA of Figure 2 is not GFM since once the agent picks q_1, the adversary can force only c loops. Note instead, adding to the NBA of Figure 2 a self loop with all letters on q_0 makes it GFM (see Figure 3), though it then recognises a different language.

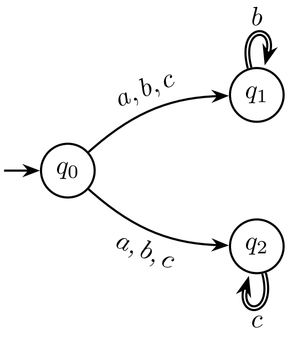

Figure 2: NBA example not good-for-MDP

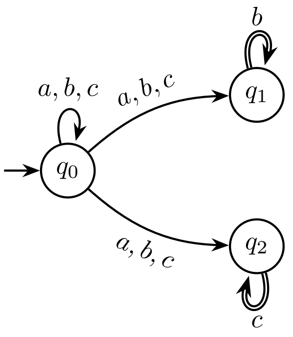

Figure 3: NBA example good-for-MDP

D Proof of Theorem 2

Proof. Obviously, by definition, $\mathsf{Psem}(\mathcal{M},\mathcal{D}) = \mathsf{Psem}(\mathcal{M},\mathcal{P})$ since $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{D})$.

Since $\mathcal{D}$ is deterministic and thus GFM, $\mathsf{Psyn}(\mathcal{M},\mathcal{D}) = \mathsf{Psem}(\mathcal{M},\mathcal{D})$. Therefore, we only need to show that $\mathsf{Psyn}(\mathcal{M},\mathcal{P}) = \mathsf{Psem}(\mathcal{M},\mathcal{D})$.

First, we prove that $\mathsf{Psem}(\mathcal{M},\mathcal{D}) \le \mathsf{Psyn}(\mathcal{M},\mathcal{P})$. Let μ be the optimal strategy for $\mathcal{M}$ to achieve $\mathsf{Psem}(\mathcal{M},\mathcal{D})$. We then can apply μ to resolve the nondeterminism in the actions and obtain a MC $\mathcal{M}^\mu$. Further, $\mathsf{Psyn}(\mathcal{M}^\mu,\mathcal{D}) = \mathsf{Psem}(\mathcal{M}^\mu,\mathcal{D}) = \mathsf{Psem}(\mathcal{M},\mathcal{D})$ since μ is an optimal strategy. This in fact gives a product $\mathcal{M}^\mu \otimes \mathcal{P}$ as the strategy μ does not rely on the 0/1-PA $\mathcal{P}$. According to (Li et al. 2025, Proposition 2), the product $\mathcal{M}^\mu \otimes \mathcal{P}$ achieves the maximal semantic probability $\mathsf{Psem}(\mathcal{M}^\mu,\mathcal{P})$, i.e., $\mathsf{Psyn}(\mathcal{M}^\mu,\mathcal{P}) = \mathsf{Psem}(\mathcal{M}^\mu,\mathcal{P})$. This entails that $\mathsf{Psem}(\mathcal{M}^\mu,\mathcal{P}) \le \mathsf{Psyn}(\mathcal{M},\mathcal{P})$. That is, we have $\mathsf{Psem}(\mathcal{M}^\mu,\mathcal{D}) = \mathsf{Psem}(\mathcal{M}^\mu,\mathcal{P}) \le \mathsf{Psyn}(\mathcal{M},\mathcal{P})$ since $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{P})$. Again, since μ is the optimal strategy and $\mathcal{M}^\mu$ is the induced MC, we have $\mathsf{Psem}(\mathcal{M}^\mu,\mathcal{D}) = \mathsf{Psem}(\mathcal{M},\mathcal{D})$. It then follows that $\mathsf{Psem}(\mathcal{M},\mathcal{D}) \le \mathsf{Psyn}(\mathcal{M},\mathcal{P})$.

We now show that $\mathsf{Psyn}(\mathcal{M},\mathcal{P}) \le \mathsf{Psem}(\mathcal{M},\mathcal{D})$. Let μ be any optimal finite memory strategy on $\mathcal{M}$ such that $\mathsf{Psyn}(\mathcal{M}^\mu,\mathcal{P}) = \mathsf{Psyn}(\mathcal{M},\mathcal{P})$. According to (Li et al. 2025, Proposition 2), $\mathcal{P}$ can preserve the semantic satisfaction probability for MCs, i.e., we then have $\mathsf{Psyn}(\mathcal{M}^\mu, \mathcal{P}) = \mathsf{Psem}(\mathcal{M}^\mu, \mathcal{P})$ since $\mathcal{M}^\mu$ is an MC. Note that our product MDP definition de-generalises to the product MC definition in (Li et al. 2025) given a strategy μ on $\mathcal{M}$. Together with the fact that $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{D})$, it holds that $\mathsf{Psyn}(\mathcal{M}^\mu, \mathcal{P}) = \mathsf{Psem}(\mathcal{M}^\mu, \mathcal{D})$ since $\mathsf{Psem}(\mathcal{M}^\mu, \mathcal{D}) = \mathsf{Psem}(\mathcal{M}^\mu, \mathcal{P})$. Thus, $\mathsf{Psyn}(\mathcal{M}^\mu, \mathcal{P}) = \mathsf{Psyn}(\mathcal{M}^\mu, \mathcal{D}) \leq \mathsf{Psem}(\mathcal{M}, \mathcal{D})$ as μ might not be the optimal strategy for $\mathcal{M}$ to achieve $\mathsf{Psem}(\mathcal{M}, \mathcal{D})$. It then immediately follows that $\mathsf{Psyn}(\mathcal{M}, \mathcal{P}) \leq \mathsf{Psem}(\mathcal{M}, \mathcal{D})$ since for all μ, we have $\mathsf{Psyn}(\mathcal{M}^\mu, \mathcal{P}) \leq \mathsf{Psem}(\mathcal{M}, \mathcal{D})$.

Therefore, $\mathsf{Psyn}(\mathcal{M}, \mathcal{P}) = \mathsf{Psem}(\mathcal{M}, \mathcal{D})$. We have thus completed the proof. $\square$

Equivalent 0/1-PA for Every DBA Notice, that every DBA can be transformed into an equivalent 0/1-PA by simply transitioning to the only successor with probability one.

E Proof of Lemma 1

Observe that $\mathcal{A}_{\text{DBA}}$ only make the nondeterminism of $\mathcal{A}_{\text{GFM}}$ explicit. While in the classical definition of $\mathcal{M} \times \mathcal{A}_{\text{GFM}}$, we can see that the action set Act is extended to Act $\times [k]$ so to resolve the nondeterminism in $\mathcal{A}$ by selecting different successor via actions. This is also in fact a way of resolving nondeterminism in $\mathcal{A}$.

In our constructed DBA $\mathcal{A}_{\text{DBA}}$, for each state $q \in Q_{dba}$ and a letter $\sigma \in \Sigma$, $q \in \delta_{\text{GFM}}(q, \sigma)$ if, and only if, there exists a unique integer $i \in [k]$ such that $q' = \delta_{\text{DBA}}(q, \langle \sigma, i \rangle)$. It is easy to see that we have $\mathsf{P}^\times_{\text{GFM}}(\langle s, q\rangle, \langle a, i\rangle, \langle s', q'\rangle) > 0$ in $\mathcal{M} \times \mathcal{A}$ if, and only if, we have $\mathsf{P}^\times_{\text{DBA}}(\langle s, q\rangle, \langle a, i, 1\rangle, \langle s', q'\rangle) > 0$ in $\mathcal{M}' \times \mathcal{A}_{\text{DBA}}$ for all s, s', q, a, where $q' = \delta_{dba}(q, \langle L(s, a), i\rangle)$. Further, there is one-to-one correspondence between the two transitions $(\langle s, q\rangle, \langle a, i\rangle, \langle s', i\rangle)$ of $\mathcal{M} \times \mathcal{A}$ and $(\langle s, q\rangle, \langle a, i, 1\rangle, \langle s', q'\rangle)$ of $\mathcal{M}' \times \mathcal{A}_{\text{DBA}}$, with $\mathsf{P}^\times_{\text{GFM}}(\langle s, q\rangle, \langle a, i\rangle, \langle s', q'\rangle) = \mathsf{P}^\times_{\text{DBA}}(\langle s, q\rangle, \langle a, i, 1\rangle, \langle s', q'\rangle) = \mathsf{P}(s, a, s')$. Since $\mathcal{A}_{\text{GFM}}$ is GFM, there is a memoryless strategy from every state in $\mathcal{M} \times \mathcal{A}_{\text{GFM}}$ to obtain the corresponding optimal satisfaction probability. Therefore, from a memoryless optimal strategy on $\mathcal{M} \times \mathcal{A}_{\text{GFM}}$, we can immediately construct an optimal strategy on $\mathcal{M}' \times \mathcal{A}_{\text{DBA}}$. On the other hand, the product $\mathcal{M}' \times \mathcal{A}_{\text{DBA}}$ is essentially the same as $\mathcal{M} \times \mathcal{A}_{\text{GFM}}$. Hence, from every strategy μ' of a state $\langle s, q\rangle$ in $\mathcal{M}' \times \mathcal{A}_{\text{DBA}}$, regardless of memoryless or not, we can naturally construct a strategy μ of the state $\langle s, q\rangle$ in $\mathcal{M} \times \mathcal{A}_{\text{GFM}}$ obtaining the same satisfaction probability.

Therefore, the lemma holds.

Optimal values coincide. By the definition of GFM automata, the optimal strategy on $\mathcal{M} \times \mathcal{A}_{\text{GFM}}$ equals the optimal satisfaction probability of φ in $\mathcal{M}$. As shown in Lemma 1, there is a one-to-one correspondence between the two product MDPs $\mathcal{M} \times \mathcal{A}_{\text{GFM}}$ and $\mathcal{M}' \times \mathcal{A}_{\text{DBA}}$, with essentially identical state spaces, transitions, and probabilities. Hence, both yield the same optimal satisfaction probability.

F Proof of Lemma 2

Lemma 2 directly follows from (Li et al. 2025, Lemma 3). In order to be self-contained, we briefly sketch the proof idea here. From a DBA $\mathcal{A}_{\text{DBA}}$ and thus a DCA $\mathcal{A}_{\text{DCA}}$, we obtain a minimal GFG NCA $\mathcal{A}_{\text{GFG-Min}}$ whose language is exactly $\mathcal{L}(\mathcal{A}_{\text{DCA}})$. $\mathcal{A}_{\text{GFG-Min}}$ are both semantically deterministic and safe deterministic. As mentioned earlier in the paper, $\mathcal{A}_{\text{GFG-Min}}$ is semantically deterministic, if for any state q and letter σ, all successors have the same language, i.e., for two states $s, t \in \delta_{\text{GFGMin}}(q, \sigma)$, we have $\mathcal{L}(\mathcal{A}^s_{\text{GFG-Min}}) = \mathcal{L}(\mathcal{A}^t_{\text{GFG-Min}})$. $\mathcal{A}_{\text{GFG-Min}}$ is also safe deterministic if when removing all α_{GFGMin}-transitions, all strongly connected components are deterministic. Since $\mathcal{A}_{\text{GFG-Min}}$ has a co-Büchi condition, every accepting run will eventually enter the so-called deterministic safe component because $\mathcal{A}_{\text{GFG-Min}}$ is safe-deterministic.

For any word $w \in \mathcal{L}(\mathcal{A}_{\text{DBA}})$, which is of course not in $\mathcal{L}(\mathcal{A}_{\text{DCA}})$ (and thus not in $\mathcal{L}(\mathcal{A}_{\text{GFG-Min}})$) as $\mathcal{A}_{\text{DBA}}$ and $\mathcal{A}_{\text{DCA}}$ complement each other, all runs in $\mathcal{A}_{\text{GFG-Min}}$ over w are not accepting. Hence, when $\mathcal{A}_{\text{GFG-Min}}$ is treated as an NBA $\mathcal{A}_{\text{NBA}}$, all runs over w are accepting. Therefore, the probability measure of the accepting runs over an accepting word w in $\mathcal{A}_{\text{PA}}$ is one. Now, we show that the probability measure of accepting runs over a rejecting word $w \notin \mathcal{L}(\mathcal{A}_{\text{DBA}})$ is zero. This is also easy as $\mathcal{A}_{\text{GFG-Min}}$ is semantically deterministic. Let $w \notin \mathcal{L}(\mathcal{A}_{\text{DBA}})$ and thus we have $w \in \mathcal{L}(\mathcal{A}_{\text{GFG-Min}})$. There must be an accepting run $q_0 q_1 q_2 \cdots$ over w in $\mathcal{A}_{\text{GFG-Min}}$. For any run $q'_0 q'_1 q'_2 \cdots$, we know that $\mathcal{L}(\mathcal{A}^{q'_i}_{\text{GFG-Min}}) = \mathcal{L}(\mathcal{A}^{q_i}_{\text{GFG-Min}})$ for all $i \geq 0$ since $\mathcal{A}_{\text{GFG-Min}}$ is semantically deterministic. Therefore, every state q'_i has a finite path to a deterministic safe component since from q_i, the remaining suffix of w will be accepted. Hence, it is always possible for a nonaccepting run in $\mathcal{A}_{\text{GFG-Min}}$ to be changed to an accepting run at any time for a positive probability. That is, every run in $\mathcal{A}_{\text{NBA}}$ over $w \notin \mathcal{L}(\mathcal{A}_{\text{DBA}})$ has a positive probability to reach a safe deterministic component at any time. So, the probability measure of accepting runs over $w \notin \mathcal{L}(\mathcal{A}_{\text{DBA}})$ must be zero since it is always with positive probability to escape to nonaccepting zone in $\mathcal{A}_{\text{NBA}}$.

It then follows that the $\mathcal{A}_{\text{PA}}$ is 0/1-PA and recognises the language $\mathcal{L}(\mathcal{A}_{\text{DBA}})$.

G Preservation of ω-Regular Properties under GFG minimisation

GFM automata recognise all ω-regular languages (Hahn et al. 2020). In our reduction pipeline, we first transform these GFM automata into DBAs by making the nondeterministic choices explicit, using additional letters and then transform them into 0/1-PAs. Note that the original ω-regular language can always be recovered by simply ignoring these extra letters. Hence, the ω-regular properties remain fully preserved throughout our reduction.

H Proof of Theorem 4

Proof. As a trivial result of Theorem 1, within an accepting MEC, there is *memoryless* and random strategy to visit all state-actions pairs with probability one. Therefore, we can treat all accepting MECs as sink goal states. Since there exists a *memoryless* optimal strategy for reachability goals, we have a memoryless strategy on $\mathcal{M}' \otimes \mathcal{P}$ to solve the stochastic planning problem. From the optimal strategy $\mu_{\mathcal{M}'}$ for $\mathcal{M}' \otimes \mathcal{P}$, we can obtain the optimal strategy for $\mathcal{M}'$ to achieve $\mathsf{Psem}(\mathcal{M}', \mathcal{P})$. Further, from an optimal strategy $\mu_{\mathcal{M}'} : S \times Q_{\mathcal{P}} \to \mathsf{Distr}(\mathrm{Act}')$, we can obtain an optimal strategy $\mu_{\mathcal{M}} : S \times Q_{\mathcal{P}} \to \mathsf{Distr}(\mathrm{Act})$ by setting $\mu_{\mathcal{M}}(\langle s, q\rangle)(a) = \Sigma_{i\in[k]}\mu_{\mathcal{M}'}(\langle s, q\rangle)(\langle a, i\rangle)$ to achieve $\mathsf{Psem}(\mathcal{M}', \mathcal{A}_{\mathrm{DBA}}) = \mathsf{Psem}(\mathcal{M}', \mathcal{P})$. Further, by Lemma 1, we know that a strategy that maximises the satisfaction probability of $\mathcal{L}(\mathcal{A}_{\mathrm{DBA}})$ from the initial state in $\mathcal{M}'$ is also a strategy that maximises the satisfaction probability of $\mathcal{L}(\mathcal{A})$ in the same initial state in $\mathcal{M}$. Since $\mathsf{Psem}(\mathcal{M}, \mathcal{A}) = \mathsf{Psem}(\mathcal{M}', \mathcal{A}_{\mathrm{DBA}})$, we then have obtained the optimal strategy by letting $\mu' = \mu_{\mathcal{M}}$. □

I Proof Lemma 3

Proof. It is easy to see that the right-hand side implies the left-hand side. If P is specifiable in $\mathsf{GF}\varphi$ where φ is a co-safety formula, then for any $w \in P$, we have that $w_i \models \varphi$ for infinitely many integers $i > 0$. Let $w[i \cdots i'] \models \varphi$ for infinitely many i and i' with $i \leq i'$. We can let $[\varphi] = R \cdot tt^{\omega}$. Then, we know that for any $w \in P$, there are infinitely many prefixes of w belong to $\Sigma^* \cdot R$. Therefore, P is a repeated reachability property specifiable in LTL.

The implication from the left to the right is an immediate result from (Sickert and Esparza 2020, Theorem 3). Note that the strong release temporal operator M can be rewritten as $\varphi \mathsf{M} \psi \equiv \psi \mathsf{U} (\varphi \wedge \psi)$. So, any co-safety/reachability/guarantee property in R can be specified by a co-safety formula φ. To make it repeat infinitely often as prefixes, we can just add GF in front of φ. □

J Proof of Theorem 5

Proof. Let φ be a G-free formula, i.e., a formula that does not contain G modality and w be a word. Then $w \models \varphi$ if, and only if, there exists an integer $j > 0$, $af(\varphi, w[0 \cdots j]) = tt$, according to (Esparza and Kretínský 2014, Lemma 11). The function af is defined as in (Esparza and Kretínský 2014).

Recall that for a co-safety property formula φ, one can construct an NFA $\mathcal{N}_{\varphi} = (Q, q_0, \delta, F)$ such that $\mathcal{L}_*(\mathcal{N}_{\varphi}) \cdot (tt)^{\omega} = [\varphi]$ where $\mathcal{L}_*(\mathcal{N})$ denotes the set of finite traces accepted by $\mathcal{N}_{\varphi}$. Note that the alphabet here is $\Sigma = 2^{\mathsf{AP}}$.

Recall now the construction for the GFM automaton $\mathcal{A}$ from $\mathcal{N}_{\varphi}$. We construct from $\mathcal{N}_{\varphi}$ a GFM automaton $\mathcal{A} = (Q_{\mathcal{A}}, q_{\mathcal{A}}, \delta_{\mathcal{A}}, \alpha_{\mathcal{A}})$ for $\mathsf{GF}\varphi$ where

- $Q_{\mathcal{A}} \subseteq ((Q \setminus F) \cup \{q_0\})$, $q_{\mathcal{A}} = q_0$,
- $\delta_{\mathcal{A}} : Q_{\mathcal{A}} \times \Sigma \to 2^{Q_{\mathcal{A}}}$ is defined such that for each $q \in Q_{\mathcal{A}}$ and $\sigma \in \Sigma$, we have (1) $q' \in \delta_{\mathcal{A}}(q, \sigma)$ if $q' \in \delta(q, \sigma)$ with $q' \notin F$, and (2) $q_0 \in \delta_{\mathcal{A}}(q, \sigma)$, and
- $\alpha_{\mathcal{A}} = \{(q, \sigma, q_0) \in \Delta(\delta_{\mathcal{A}}) \mid \exists q' \in \delta(q, \sigma) \wedge q' \in F\}$.

We first prove that $\mathcal{L}(\mathcal{A}) = [\mathsf{GF}\varphi]$. Assume that $w \models \mathsf{GF}\varphi$. By definition, for any given integer $i > 0$, there exists a position $j \geq i$ such that $w_j \models \varphi$. Assume that $\mathbf{J} = \{j > 0 \mid w_j \models \varphi\} = \{j_0, j_1, \cdots\}$. We can construct a strategy to produce an accepting run ρ for w in $\mathcal{A}$ as follows: before reaching position j_0, we choose to stay in the initial state q_0. Since $w_{j_0} \models \varphi$, there must exist a position $j_0' \geq j_0$ such that $af(\varphi, w[j_0 \cdots j_0']) = tt$. So, we know that $\delta(q_0, w[j_0 \cdots j_0']) \cap F \neq \emptyset$, i.e., there must be an accepting finite run to a final state, say f_0. By construction, we are able to transition back to q_0 when we read the word $w[j_0 \cdots j_0']$ instead of reaching the final state f_0. The corresponding transition back to q_0 is accepting and in $\alpha_{\mathcal{A}}$. Next, we select the smallest integer $j_k \in \mathbf{J}$ such that $j_k > j_0'$. We choose to loop on the initial state q_0 until we reach position j_k and then move according to the δ function afterwards. Analogously, we are able to go back to the initial state q_0 via an accepting transition because $w_{j_k} \models \varphi$. If we repeat this strategy, in the end, the constructed run ρ will visit accepting transitions for infinitely often. Therefore, $w \in \mathcal{L}(\mathcal{A})$. That is, we have $[\mathsf{GF}\varphi] \subseteq \mathcal{L}(\mathcal{A})$.

The proof for the other direction that $\mathcal{L}(\mathcal{A}) \subseteq [\mathsf{GF}\varphi]$ is straightforward. This is because as soon as the accepting transition is visited, the finite word from the latest initial state to the accepting transition must satisfy φ. Therefore, if an accepting run visits accepting transitions for infinitely many times, the corresponding word also satisfies φ at infinitely many positions.

It then follows that $\mathcal{L}(\mathcal{A}) = [\mathsf{GF}\varphi]$.

Now we have to prove that $\mathcal{A}$ is GFM. We can actually prove a stronger result that $\mathcal{A}$ is stochastically resolvable automaton in a sense that we can resolve the nondeterminism randomly and obtain a language equivalent 0/1-PA $\mathcal{P}$. This follows almost directly from our construction.

We prove that $\mathbb{P}_{\mathcal{P}}(w) = 1$ for each word $w \in \mathcal{L}(\mathcal{P})$ and $\mathbb{P}_{\mathcal{P}}(w) = 0$ for $w \notin \mathcal{L}(\mathcal{A})$. Let $w \in \mathcal{L}(\mathcal{A})$. The whole transition graph of $\mathcal{A}/\mathcal{P}$ is a strongly connected component. Therefore, every finite path will be visited almost surely because the nondeterminism is resolved randomly. It then follows that all transitions will be visited infinitely often. This then entails that a run of $\mathcal{P}$ over w almost surely visits an accepting transition. Hence, $\mathbb{P}_{\mathcal{P}}(w) = 1$. Let $w \notin \mathcal{L}(\mathcal{A})$. By semantics, we know that there exists an integer $k > 0$ such that for all $j \geq k$, we have that $w_j \models \neg\varphi$. This means that every run of $\mathcal{P}$ over w must not visit the accepting transition from some point on, which would only happen with probability zero. Therefore, we have $\mathbb{P}_{\mathcal{P}}(w) = 0$ for a word $w \notin \mathcal{L}(\mathcal{A})$.

It then follows that $\mathcal{P}$ is a language-equivalent 0/1-PA of $\mathcal{A}$. Therefore, $\mathcal{A}$ is GFM according to our previous algorithm.

□

K Need for Co-Safety in $\mathsf{GF}\varphi$

For repeated reachability formulas $\mathsf{GF}\varphi$, φ are co-safety formulas that go beyond the intersection of safety and co-safety e.g. $(a\mathsf{U}b)$ (Esparza, Kretínský, and Sickert 2020). The key property we exploit of these formulas is that they can be recognised by a reachability automaton (NFA in our proof). Any specification that has this characteristic can be handled by our method.

L Optimal Strategy and Further Optimisations

We give the following Algorithm 1 to synthesise the optimal strategy to achieve $\mathsf{Psem}(\mathcal{M}', \mathcal{A}_{\mathrm{DBA}})$ in more details.

Algorithm 1: Synthesising an Optimal Strategy for an MDP $\mathcal{M}$ against an LTL specification φ

```
1: Construct a GFM automaton $\mathcal{A}$ from $\varphi$.
2: Create $\mathcal{M}'$ from $\mathcal{M}$ and $\mathcal{A}_{\text{DBA}}$ (and thus $\mathcal{A}_{\text{DCA}}$) from $\mathcal{A}$.
3: Apply GFG minimisation on $\mathcal{A}_{\text{DCA}}$ and treat the minimised automaton $\mathcal{A}_{\text{GFG-Min}}$ as a Büchi automaton $\mathcal{A}_{\text{NBA}}$.
4: Resolve the nondeterministic choices in $\mathcal{A}_{\text{NBA}}$ with random choices, yielding a 0/1-PA $\mathcal{A}_{\text{PA}}$.
4: Create $\mathcal{M}' \otimes \mathcal{A}_{\text{PA}}$.
5: Identify the MECs with accepting transitions in the product MDP and mark the states in accepting MECs as accepting states, i.e., reducing it to reachability goals.
6: Determine the chance of reaching an accepting state in $\mathcal{M}' \otimes \mathcal{A}_{\text{PA}}$, recording a strategy for those states that reach it with a probability in $(0, 1]$, i.e., solving reachability planning problems.
7: For the states in the accepting MECs, use the strategy to maximally randomise among all successors that stay within the accepting region, i.e., using a memoryless strategy to visit all transitions in the accepting MECs.
```

We can optimise the quantitative analysis after finding the accepting MECs (Step 3 of Algorithm 1) by exploiting that, for the 0/1-PA $\mathcal{A}_{\text{PA}}$, the update of the right language of states is deterministic. (Recall that $\mathcal{A}_{\text{PA}}$ is semantically deterministic.) The right language is the language accepted from that state in $\mathcal{A}_{\text{PA}}$. We call the deterministic update automata of these languages $\mathcal{R}$ and note that $\mathcal{R}$ can be viewed as a quotient automaton of $\mathcal{A}_{\text{PA}}$.

We can simply make every state (s, r) in $\mathcal{M}' \times \mathcal{R}$ accepting if there is a state (s, p) in an accepting MEC of $\mathcal{M}' \otimes \mathcal{A}_{\text{PA}}$ such that p and r are language equivalent in $\mathcal{A}_{\text{PA}}$, and then calculating the chance of winning for the remaining states by maximising the chance of reaching any such state (s, r).

This provides an optimal strategy which operates on $\mathcal{M}' \times \mathcal{R}$ until such a state (s, r) is reached, and then moves to a random strategy on $\mathcal{M}' \otimes \mathcal{A}_{\text{PA}}$ on such an accepting MEC.

M Automata Completion and Sink States

We deliberately refrained from completing all automata returned by `Owl`. This ensures that our setup remains fully reproducible and consistent with how the original tools provide their outputs. Completing them would only add one single (sink) state, without adding meaningful insight. Therefore, we keep the original automaton to maintain transparency and reproducibility.

N Additional Experiments

Tables 3 and 4 show the effectiveness of our GFM state-space reduction on additional LTL formulas, extracted from Literature. Table 5 demonstrates the applicability of our direct construction for the $\mathsf{GF}\varphi$ fragment. Where applicable, we use `GENLTL` of Spot (Duret-Lutz 2013) and existing LTL datasets from the Owl (Kretínský, Meggendorfer, and Sickert 2018) repository.

We use a sideway layout to present each LTL formula in full, together with the number of automata states. Using the same notation as in the main-body, we denote Owl (Kretínský, Meggendorfer, and Sickert 2018) by `Owl`, and the "slim" GFM construction of (Hahn et al. 2020) by `Slim`. `Owl-Red` and `Slim-Red` denote the subsequent application of our reduction. `GFM-GF` denotes our direct construction. We measure runtime in seconds and report a timeout after 300 seconds. Our reduction returns complete automata with an explicit sink state, which can add one extra state.

Furthermore, we introduce following novel pattern:

Nested Conjunction Sequence (NCS). Conceptually inspired by (Müller and Sickert 2017), we define $\mathsf{NCS}[k_1, \ldots, k_n] = \mathsf{GF}(a \wedge \mathsf{X}^{k_1} b \wedge \mathsf{X}^{k_1+k_2} c \wedge \mathsf{X}^{k_1+k_2+k_3} d \wedge \mathsf{X}^{k_1+k_2+k_3+k_4} e \wedge \ldots)$, where X^n denotes n nested applications of the next operator.

Reference	Formula	Owl	Owl-Red	Slim	Slim-Red
(Dwyer, Avrunin, and Corbett 1998)	`G(!a|G!b|((!c|X(!bU(d&Fe))|X(bR!d))Ub))`	23 (0.46)	17 (0.01)	21 (0.09)	**16** (0.01)
	`G(!a|((!b|X(!cU(d&Fe))|X(cR!d))U(c|G(!b|X(!cU(d&Fe))|X(cR!d)))))`	39 (0.47)	23 (0.03)	20 (0.09)	**15** (0.01)
(Etessami and Holzmann 2000)	`aU(b&X(c&F(d&XF(e&XF(f&XFg)))))`	13 (0.43)	**9** (0.01)	127 (0.14)	54 (0.31)
(Tabakov, Rozier, and Vardi 2012)	`G(p1->(p1U(p2&(p2U(p3&(p3U(p4&(p4Up5))))))))`	44 (1.16)	**23** (0.02)	51 (0.22)	39 (0.02)
	`G(p1->(p1U(p2&(p2U(p3&(p3U(p4&(p4U(p5&(p5Up6))))))))))`	150 (1.41)	**56** (0.17)	232 (0.82)	159 (0.76)
	`G(p1->(p1U(p2&(p2U(p3&(p3U(p4&(p4U(p5&(p5U(p6&(p6Up7))))))))))))`	433 (5.92)	**249** (27.75)	1425 (12.24)	753 (163.32)
(Duret-Lutz 2013)	`G(!a|(aU(b&(bU(c&(cUd))))))`	14 (0.48)	**10** (0.01)	12 (0.09)	13 (0.01)
	`(FGa|GFb)&(FGc|GFa)&(FGd|GFc)`	14 (0.43)	**13** (0.01)	78 (0.11)	69 (0.06)
	`(FGa|GFb)&(FGc|GFa)&(FGd|GFc)&(FGe|GFd)`	30 (0.44)	**27** (0.01)	406 (0.51)	370 (5.54)
	`GF((a&XXXa)|(!a&XXX!a))`	17 (0.43)	18 (0.01)	17 (0.09)	**10** (0.01)
	`GF((a&XXXXa)|(!a&XXXX!a))`	33 (0.44)	34 (0.01)	33 (0.09)	**18** (0.01)
	`a&X(b|c)&(!bU(b&X(c&XXXXGa)))&G(!c|X((d|e)&Xf))&G(!f|X((d|e)&X(a&X(b|c|Ga))))&F(a&X(!a&(((!c|X((d&F(b&F(c&Xd)))|(e&F(b&F(c&Xe)))))&(!f|X((d&F(b&F(f&Xd)))|(e& F(b&F(f&Xe))))))Ua)))&G((!c|!d)&(!a|!b)&(!a|!c)&(!a|!d)&(!b|!c)&(!b|!d)&(!d|!e)& (!a|!e)&(!b|!e)&(!c|!e)&(!d|!f)&(!e|!f)&(!a|!f)&(!b|!f)&(!c|!f))`	163 (140.13)	**148** (0.02)	149 (0.17)	**148** (0.01)
	`G((!c|!d)&(!a|!b)&(!a|!c)&(!a|!d)&(!b|!c)&(!b|!d))&(!aU(a&X((b|c)&XGd)))&F(d&XXd &((Xb&G(!a|Xb))|(Xc&G(!a|Xc))))`	19 (0.60)	17 (0.01)	20 (0.09)	**16** (0.01)
	`G((!c|!d)&(!a|!b)&(!a|!c)&(!a|!d)&(!b|!c)&(!b|!d))&(!bU(b&X((c|d)&X((c|d)&XGa)))) &F(a&((Xc&G(!b|Xc))|(Xd&G(!b|Xd)))&XXXa&((XXc&G(!b|XXc))|(XXd&G(!b|XXd))))`	117 (4.84)	108 (0.02)	147 (0.11)	**107** (0.02)
(Geldenhuys and Hansen 2006)	`(Fp1|Gp2)&(Fp2|Gp3)&(Fp3|Gp4)&(Fp4|Gp5)&(Fp5|Gp6)&(Fp6|Gp7)&(Fp7|Gp8)`	–	–	1690 (3.17)	**612** (204.61)
	`(GFp1|FGp2)&(GFp2|FGp3)&(GFp3|FGp4)&(GFp4|FGp5)`	30 (0.46)	**27** (0.02)	406 (0.54)	370 (5.70)
	`(GFp1|FGp2)&(GFp2|FGp3)&(GFp3|FGp4)&(GFp4|FGp5)&(GFp5|FGp6)`	66 (0.49)	**58** (0.03)	3074 (9.09)	timeout
	`(GFp1|FGp2)&(GFp2|FGp3)&(GFp3|FGp4)&(GFp4|FGp5)&(GFp5|FGp6)&(GFp6|FGp7)`	146 (1.01)	**127** (0.14)	timeout	–
	`(GFp1|FGp2)&(GFp2|FGp3)&(GFp3|FGp4)&(GFp4|FGp5)&(GFp5|FGp6)&(GFp6|FGp7)&(GFp7|FGp8)`	322 (3.78)	**279** (0.96)	timeout	–
	`(GFp1|FGp2)&(GFp2|FGp3)&(GFp3|FGp4)&(GFp4|FGp5)&(GFp5|FGp6)&(GFp6|FGp7)&(GFp7|FGp8) &(GFp8|FGp9)`	706 (23.72)	**612** (8.78)	timeout	–
	`(GFp1|FGp2)&(GFp2|FGp3)&(GFp3|FGp4)&(GFp4|FGp5)&(GFp5|FGp6)&(GFp6|FGp7)&(GFp7|FGp8) &(GFp8|FGp9)&(GFp9|FGp10)`	1538 (226.73)	**1337** (76.20)	timeout	–
(Müller and Sickert 2017)	`F(b1&F(b2&F(b3&Fb4)))&GF(a1&X(a2&X(a3&Xa4)))`	13 (0.74)	14 (0.01)	14 (0.16)	**10** (0.01)
	`F(b1&F(b2&F(b3&F(b4&Fb5))))&GF(a1&X(a2&X(a3&X(a4&Xa5))))`	22 (0.76)	23 (0.04)	23 (0.19)	**14** (0.03)
	`F(b1&F(b2&F(b3&F(b4&F(b5&Fb6)))))&GF(a1&X(a2&X(a3&X(a4&X(a5&Xa6)))))`	39 (0.77)	40 (0.27)	40 (0.44)	**21** (0.19)
	`F(b1&F(b2&F(b3&F(b4&F(b5&F(b6&Fb7))))))&GF(a1&X(a2&X(a3&X(a4&X(a5&X(a6&Xa7))) )))`	72 (0.92)	73 (2.99)	73 (1.57)	**34** (1.76)
	`F(b1&F(b2&F(b3&F(b4&F(b5&F(b6&F(b7&Fb8)))))))&GF(a1&X(a2&X(a3&X(a4&X(a5&X(a6 &X(a7&Xa8)))))))`	137 (1.45)	138 (39.11)	138 (10.74)	**59** (20.84)

Table 3: Comparison of automata state spaces of `Owl` and `Slim` against their reduced versions, `Owl-Red` and `Slim-Red`. We report number of states (smallest in bold) and runtime (in seconds). Within `Owl-Red` and `Slim-Red` we only measure the time it took to reduce the automaton. We report timeout after 300 seconds. The reduction returns complete automata, with explicit sink state, which is why for some cases, the reduction increases the state size by one.

Reference	Formula	`Owl`	`Owl-Red`	`Slim`	`Slim-Red`
(Holeček et al. 2004)	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3))`	13 (0.44)	**8** (0.01)	17 (0.10)	10 (0.01)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4))`	17 (0.44)	**10** (0.01)	33 (0.11)	18 (0.13)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5))`	21 (0.48)	**12** (0.01)	65 (0.20)	34 (3.83)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5)|(a6&X!a6)|(!a6&Xa6))`	25 (0.62)	**14** (0.01)	129 (0.69)	66 (128.34)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5)|(a6&X!a6)|(!a6&Xa6)|(a7&X!a7)|(!a7&Xa7))`	29 (1.45)	**16** (0.01)	257 (3.70)	timeout
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5)|(a6&X!a6)|(!a6&Xa6)|(a7&X!a7)|(!a7&Xa7)|(a8&X!a8)|(!a8&Xa8))`	33 (5.93)	**18** (0.02)	513 (27.97)	timeout
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5)|(a6&X!a6)|(!a6&Xa6)|(a7&X!a7)|(!a7&Xa7)|(a8&X!a8)|(!a8&Xa8)|(a9&X!a9)|(!a9&Xa9))`	37 (37.75)	**20** (0.03)	timeout	–
(Baier et al. 2023)	`G("msgsend"->F("acksend"&(("acKrev")|X(("acKrev")|X(("acKrev")|X("acKrev"))))))`	9 (0.49)	**7** (0.01)	37 (0.10)	16 (0.01)
	`G("msgsend"->F("acksend"&(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X("acKrev")))))))`	19 (0.50)	**9** (0.01)	77 (0.12)	27 (0.02)
	`G("msgsend"->F("acksend"&(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X("acKrev"))))))))`	37 (0.51)	**15** (0.01)	157 (0.11)	41 (0.05)
	`G("msgsend"->F("acksend"&(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X("acKrev")))))))))`	69 (0.55)	**17** (0.02)	317 (0.14)	65 (0.17)
	`G("msgsend"->F("acksend"&(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X("acKrev"))))))))))`	133 (0.65)	**19** (0.04)	637 (0.21)	95 (0.71)
	`G("msgsend"->F("acksend"&(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X("acKrev")))))))))))`	261 (0.86)	**21** (0.14)	1277 (0.33)	146 (3.49)
	`G("msgsend"->F("acksend"&(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X("acKrev"))))))))))))`	517 (1.51)	**24** (0.60)	2557 (0.64)	202 (19.72)
	`G("msgsend"->F("acksend"&(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X(("acKrev")|X("acKrev")))))))))))))`	1029 (3.04)	**25** (2.25)	5117 (1.28)	283 (97.48)
NCS	`GF(a&X(b)&X(X(X(c))))`	8 (0.46)	8 (0.01)	9 (0.09)	**6** (0.01)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(d)))))`	16 (0.46)	16 (0.01)	17 (0.09)	**10** (0.01)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(d))))))`	32 (0.46)	32 (0.03)	33 (0.09)	**18** (0.01)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d)))))))`	64 (0.46)	64 (0.12)	65 (0.10)	**34** (0.02)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d))))))&X(X(X(X(X(X(X(e))))))))`	128 (0.52)	128 (1.48)	129 (0.14)	**66** (0.05)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d))))))&X(X(X(X(X(X(X(X(e)))))))))`	256 (0.62)	256 (10.96)	257 (0.22)	**130** (0.15)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d))))))&X(X(X(X(X(X(X(X(X(e))))))))))`	512 (0.71)	512 (83.81)	513 (0.42)	**258** (0.55)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d))))))&X(X(X(X(X(X(X(X(X(X(e)))))))))))`	1024 (1.06)	timeout	1025 (0.53)	**514** (2.31)

Table 4: Comparison of automata state spaces of `Owl` and `Slim` against their reduced versions, `Owl-Red` and `Slim-Red`. We report number of states (smallest in bold) and runtime (in seconds). Within `Owl-Red` and `Slim-Red` we only measure the time it took to reduce the automaton. We report timeout after 300 seconds.

Reference	Formula	`Owl-Red`	`Slim-Red`	`GFM-GF`
(Holeček et al. 2004)	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3))`	8 (0.45)	10 (0.11)	**7** (0.09)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4))`	10 (0.45)	18 (0.23)	**9** (0.09)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5))`	12 (0.49)	34 (4.03)	**11** (0.09)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5)|(a6&X!a6)|(!a6&Xa6))`	14 (0.63)	66 (129.03)	**13** (0.09)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5)|(a6&X!a6)|(!a6&Xa6)|(a7&X!a7)|(!a7&Xa7))`	16 (1.47)	timeout	**15** (0.12)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5)|(a6&X!a6)|(!a6&Xa6)|(a7&X!a7)|(!a7&Xa7)|(a8&X!a8)|(!a8&Xa8))`	18 (5.95)	timeout	**17** (0.20)
	`GF((a1&X!a1)|(!a1&Xa1)|(a2&X!a2)|(!a2&Xa2)|(a3&X!a3)|(!a3&Xa3)|(a4&X!a4)|(!a4&Xa4)|(a5&X!a5)|(!a5&Xa5)|(a6&X!a6)|(!a6&Xa6)|(a7&X!a7)|(!a7&Xa7)|(a8&X!a8)|(!a8&Xa8)|(a9&X!a9)|(!a9&Xa9))`	20 (37.78)	timeout	**19** (0.23)
(Duret-Lutz 2013)	`GF((a&XXXa)|(!a&XXX!a))`	18 (0.44)	10 (0.09)	**7** (0.08)
	`GF((a&XXXXa)|(!a&XXXX!a))`	34 (0.45)	18 (0.10)	**9** (0.09)
	`GF(a|b)`	4 (0.42)	**1** (0.09)	**1** (0.08)
TDR	`GF(a&X(X(X(b))))`	8 (0.47)	6 (0.11)	**4** (0.09)
	`GF(a&X(X(X(X(b)))))`	16 (0.47)	10 (0.10)	**5** (0.08)
	`GF(a&X(X(X(X(X(b))))))`	32 (0.48)	18 (0.10)	**6** (0.09)
	`GF(a&X(X(X(X(X(X(b)))))))`	64 (0.49)	34 (0.10)	**7** (0.08)
	`GF(a&X(X(X(X(X(X(X(b))))))))`	128 (0.61)	66 (0.14)	**8** (0.08)
	`GF(a&X(X(X(X(X(X(X(X(b)))))))))`	256 (1.17)	130 (0.19)	**9** (0.08)
	`GF(a&X(X(X(X(X(X(X(X(X(b))))))))))`	512 (4.27)	258 (0.48)	**10** (0.10)
	`GF(a&X(X(X(X(X(X(X(X(X(X(b)))))))))))`	1024 (23.67)	514 (0.91)	**11** (0.09)
NCS	`GF(a&X(b)&X(X(X(c))))`	8 (0.47)	6 (0.10)	**4** (0.09)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(d)))))`	16 (0.47)	10 (0.10)	**5** (0.08)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(d))))))`	32 (0.49)	18 (0.10)	**6** (0.08)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d)))))))`	64 (0.58)	34 (0.11)	**7** (0.08)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d))))))&X(X(X(X(X(X(X(e))))))))`	128 (2.00)	66 (0.19)	**8** (0.09)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d))))))&X(X(X(X(X(X(X(X(e)))))))))`	256 (11.58)	130 (0.37)	**9** (0.08)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d))))))&X(X(X(X(X(X(X(X(X(e))))))))))`	512 (84.52)	258 (0.97)	**10** (0.08)
	`GF(a&X(b)&X(X(X(c)))&X(X(X(X(X(X(d))))))&X(X(X(X(X(X(X(X(X(X(e)))))))))))`	timeout	514 (2.85)	**11** (0.09)

Table 5: Comparison of automata state spaces of our reduced automata `Owl-Red` and `Slim-Red` against our direct construction `GFM-GF`. We report number of states (smallest in bold) and runtime (in seconds). Within `Owl-Red` and `Slim-Red` we report the total time including construction and reduction. We report timeout after 300 seconds.