Follow the STARs: Dynamic ω-Regular Shielding of Learned Probabilistic Policies

Ashwani Anand
MPI-SWS
Kaiserslautern, Germany
ashwani@mpi-sws.org

Satya Prakash Nayak
MPI-SWS
Kaiserslautern, Germany
sanayak@mpi-sws.org

Ritam Raha
MPI-SWS
Kaiserslautern, Germany
rraha@mpi-sws.org

Anne-Kathrin Schmuck
MPI-SWS
Kaiserslautern, Germany
akschmuck@mpi-sws.org

ABSTRACT

This paper presents a novel dynamic post-shielding framework that enforces the full class of ω-regular correctness properties over learned probabilistic policies. This constitutes a paradigm shift from the predominant setting of safety-shielding – i.e., ensuring that nothing bad ever happens – to a shielding process that additionally enforces liveness – i.e., ensures that something good eventually happens. At the core, our method uses *Strategy-Template-based Adaptive Runtime Shields (STARs)*, which leverage permissive strategy templates to enable post-shielding with minimal interference. As its main feature, STARs introduce a mechanism to *dynamically control interference*, allowing a tunable enforcement parameter to balance formal obligations and task-specific behavior *at runtime*. This allows triggering more aggressive enforcement when needed while allowing for optimized policy choices otherwise. In addition, STARs support runtime adaptation to changing specifications or actuator failures, making them especially suited for cyber-physical applications. We evaluate STARs on various benchmarks to showcase their scalability, adaptability, and performance.

KEYWORDS

Shielding; *omega*-regular Games; Strategy Templates; Reinforcement Learning; Run-time Monitoring

ACM Reference Format:
Ashwani Anand, Satya Prakash Nayak, Ritam Raha, and Anne-Kathrin Schmuck. 2026. Follow the STARs: Dynamic ω-Regular Shielding of Learned Probabilistic Policies. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/WLFP2035

1 INTRODUCTION

Motivation. Adhering to formal correctness while simultaneously optimizing performance is a core challenge in the design of autonomous cyber-physical systems (CPS) [15, 37]. While machine learning techniques – especially reinforcement learning – are highly effective at generating policies optimizing quantitative rewards, they often struggle to enforce qualitative correctness criteria such as safety or liveness. In contrast, formal methods excel at specifying and verifying such qualitative properties, typically using temporal logic or automata. Integrating these strengths has been a growing focus in safe and explainable autonomy and has led to a rich body of work integrating logical specifications into policy synthesis via multi-objective formulations [12, 13, 23, 36], or into policy reinforcement learning through automata-based reward shaping [10, 19, 20, 25, 28, 42]. While these approaches can yield policies that satisfy complex goals while adhering to formal specifications, they incorporate the specification into the synthesis or learning procedures – requiring re-training when formal objectives, environment conditions, or reward structures change.

Shielding. A promising alternative to retraining is shielding[1] – a lightweight, runtime approach that monitors and, if necessary, overrides the actions proposed by a policy to maintain compliance with a formal specification. Shields treat existing policies as a black box and ensure correctness in a *minimally interfering* manner i.e., they intervene if *and only if* the systems executions will (surely) violate the specification (in the future). The concept of shielding traces back to the foundational works on runtime monitoring of program executions in computer science [21, 29], and formal supervision of feedback control software in engineering [34]. More recently, several shielding frameworks tailored for learned policies in autonomous CPS have been introduced (see surveys by Hsu et al. [22], Könighofer et al. [26], Odriozola-Olalde et al. [30], Wabersich et al. [41]). Crucially, because shielding operates at runtime, it allows goals, safety regions, or constraints to be updated online, replacing retraining with a re-computation of the shield which often requires significantly less computational resources.

Challenges. The primary challenge in shielding is to ensure correctness and minimal interference – with only black-box runtime access to the (learned) nominal policy. Prior frameworks have primarily focused on safety, where synthesizing maximally permissive shields (which are 'inherently' minimally interfering) is tractable [2, 27, 35]. In contrast to safety, shielding for liveness is challenging because such specifications (i) do not easily allow for permissive strategies, and (ii) only manifest themselves in the infinite limit, hardening their enforcement at runtime given only a

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/WLFP2035

[1] While there are pre-shielding frameworks that apply shielding during training, we focus solely on post-shielding at runtime.

finite prefix. However, the need for shields which enforce the full class of ω-*regular specifications* naturally arises in CPS applications. A natural example are warehouse robots which should not drop a parcel (safety) but should also deliver it eventually (liveness) [17].

Contribution. To close this gap, this paper presents a *dynamic* shielding framework – *Strategy Templates based Adaptive Runtime Shields (STARs)* – which enforces *the full class of* ω*-regular specifications* over learned probabilistic policies. In addition to being correct and minimal interfering, our shields are equipped with an *enforcement parameter* γ. This enables runtime adjustment of specification enforcement across contexts: increase γ when liveness is urgent (e.g., reaching a safe zone after an alarm), and keep γ low otherwise to leverage mission-specific performance objectives optimized during policy learning.

Key Idea. The core idea behind the STARs framework is to utilize *strategy templates* recently introduced by Anand et al. [5] as their main building block. Strategy templates offer an alternative representation of strategies in two-player parity games (resulting from an ω-regular specification) that condense an infinite number of winning strategies into a simple and efficiently computable data structure. This makes them (i) truly permissive, enabling minimal inference shielding, and (ii) localizes required future progress (over a known transition graph[2]), enabling a purely history-induced evaluation of liveness properties. Thereby, strategy templates constitute the "missing piece" that makes liveness shielding tractable. In addition, strategy templates are inherently robust and composable, allowing STARs to handle dynamic runtime changes, such as unavailable actions or shifting goals.

1.1 Applications

FactoryBot is a novel benchmark, which simulates multiple *gridworld environments* in OpenAI Gym [8] such as frozen lake, taxi or cliff walking, typically used to evaluate RL policies and represents a simplified version of the snake example used to evaluate a dynamic safety shield [27]. Furthermore, adaptability in such environments, e.g., dynamic goal changes in Frozen Lake, are also subsumed by the more general capabilities of our FactoryBot setup. In FactoryBot, the nominal agent policy optimizes a reward function and STARs are used to guide the agent towards satisfying (generalized) Büchi objectives[3] which might change at runtime.

Overcooked-AI [11] is a widely used benchmark for collaborative reinforcement learning with multiple actors. Here, autonomous agents are trained to repeatedly perform cooking tasks. We use LTL specifications to encode additional recipe requirements of produced dishes. STARs are used to enforce the (additional) production of soups satisfying these requirements infinitely often.

LunarLander [9] is a standard benchmark in reinforcement learning. The simulated lunar lander can be controlled by using its directional thrusters to navigate the environment. The classical objective is to train the lander to land on a designated helipad without crashing. To demonstrate the utility of STARs, we slightly modified this benchmark to spawn the helipad at a *random position* at runtime and trained a baseline policy that just ensures that the lander lands safely — though not necessarily on the helipad. We then apply STARs to enforce landing safely *on the pad at runtime*.

Liveness Shielding. Both FactoryBot and Overcooked-AI are based on a known finite MDP. We can therefore synthesize formal shields over a game graph derived from this MDP which guarantee that the agent *always* fulfills the shielded objective (i.e. visiting the green region or producing particular soups always again) with a frequency determined by the enforcement parameter γ. On the other hand, LunarLander is based on an eight-dimensional hidden continuous MDP, which prevents us from computing a provably correct shield. Instead, we construct a very simple 2-dimensional grid-abstraction of the landers x/y position that naively assumes that each control action makes the lander move one grid cell down, right, left, or up, respectively. Even though the actual dynamics of the lander are very complex, leading to quite different abstract behavior, this simple graph allows us to compute a shield which ensures that the lander lands fast and safe on the helipad in 87% of 200 instances we simulated with the right choice of γ. This demonstrates that STARs enable the blending of quantitative objectives optimized in RL and qualitative liveness objectives at runtime, with promising future applications of STARs for deep RL shielding.

1.2 Related Work

(Post-) shielding approaches have so far focused mainly on *safety* shielding, where our work is the closest related to the works of Alshiekh et al. [2], Könighofer et al. [27], Reed et al. [35]. Similar ideas are also used for *policy repair* w.r.t. safety violations by Pathak et al. [32], Tappler et al. [38], van Waveren et al. [39], Zhou et al. [44] or via (partial) re-synthesis by Pacheck and Kress-Gazit [31], Vasilopoulos et al. [40]. In contrast, STARs directly inherit robustness and adaptability properties of strategy templates that typically circumvent the need for strategy repair and achieve necessary strategy adaptations directly via shielding. For general ω-regular specifications, our work is closely related to the runtime optimization [6] which propose a similar *blending* of a nominal policy with an additional liveness objective, however, via a very different shield synthesis technique. In contrast to our work, the shield synthesized in [6] uses a fixed enforcement parameter, does not exploit probabilistic policies, and allows no dynamic adaptation in the specification or in the graph.

This work also relates to synthesizing mean-payoff policies under ω-regular constraints [1, 43], and to pre-shielding frameworks [10, 19, 25, 28, 30, 42]. Achieving similar optimality in post-shielding is harder, but STARs can approach optimal rewards when the winning region is strongly connected. Addressing correctness without full MDP access remains a challenge for future work.

Other work that is not directly related to shielding, but is relevant to our work along with all proofs and additional formalizations can be found in the full version of the paper [4].

2 PRELIMINARIES

This section provides an overview of notation and concepts.

Notation. We denote by $\mathbb{R}$ the set of real numbers and $[a; b]$ represents the interval $\{a, a+1 \cdots, b\}$. We write Σ^* and Σ^ω to denote the

[2]We only need access to the graph *structure*, not to transition probabilities, rewards or computed policies.
[3]The choice of simple objectives is for illustration only. Our algorithm can handle the full class of ω-regular objectives.

(a) Unshielded Bot (b) Shielded Bot (c) Unshielded Lander (d) Shielded Lander

Figure 1: Applying STARs on different benchmarks. For FactoryBot: The agent must visit $\mathcal{B}$ (green), while maximizing the average reward (small numbers in cells). Images show the agent's heatmap without a shield (a), with our shield (b). For LunarLander: The agent must land on the helipad (between the flags) while avoiding crashing. Images show the lander's trajectory without a shield (c), with our shield (d).

set of finite and infinite sequences of elements from a set Σ, respectively. A *probability distribution* over a finite set S is denoted as a function $\mu : S \mapsto [0,1]$ such that $\sum_{s\in S} \mu(s) = 1$. The set of all probability distributions over S is denoted as $\mathcal{D}(S)$. The *support* of a distribution μ is the set $supp(\mu) = \{s \in S \mid \mu(s) > 0\}$. Given two distributions $\mu_1, \mu_2 \in \mathcal{D}(S)$, the *total variation distance* between μ_1 and μ_2 is defined as: $\mathrm{D_{TV}}(\mu_1, \mu_2) = \frac{1}{2}\sum_{s\in S} |\mu_1(s) - \mu_2(s)|$. Given any function $\mu : S \mapsto \mathbb{R}$, we use $\mathcal{N}(\mu) \in \mathcal{D}(S)$ to denote its normalized distribution: $\mathcal{N}(\mu)(s) = \frac{\mu'(s)}{\sum_{s'\in S}\mu'(s')}$, where $\mu'(s') = \max(\mu(s'), 0)$.

Markov Decision Process. A *Markov Decision Process* (MDP) is a tuple $M = \langle Q, A, \Delta, q_0\rangle$ where Q is a finite set of states, A is a finite set of actions, $\Delta : Q \times A \mapsto \mathcal{D}(Q)$ is a (partial) transition function and $q_0 \in Q$ is the initial state. For any state $q \in Q$, we let $A(q)$ denote the set of actions that can be selected in state q. A *strongly connected component (SCC)* of an MDP M is a maximal set of states $Q' \subseteq Q$ such that for every pair of states $q, q' \in Q'$, there exists a path from q to q' in Q' with non-zero probability.

Given a state q and an action $a \in A(q)$, we denote the probability of reaching the successor state q' from q by taking action a as $pr(q'|q,a)$. A run ρ of an MDP M is an infinite sequence in $Q \times (A \times Q)^\omega$ of the form $q_0a_0q_1\ldots$ such that $pr(q_{i+1}|q_i,a_i) > 0$. A *finite run* of *length* n is a finite such sequence $\kappa = q_0a_0\ldots q_na_n$ or $\kappa = q_0a_0\ldots q_n$. We write $\rho[i]$ to denote the i^{th} state-action pair (q_i,a_i) appearing in ρ, $\rho[i;j]$ to denote the infix $q_ia_i\ldots q_j$ for $j \geq i$, and $\rho[j;\infty]$ to denote the suffix $q_ja_j\ldots$. These notations extend to the case of finite runs analogously. We write Runs^M (resp. FRuns^M) to denote the set of all infinite (resp. finite) runs of M. We denote the last state of a finite run ρ as $last(\rho)$.

A *policy* (or, strategy) in an MDP M is a function $\sigma : \mathrm{FRuns}^M \mapsto \mathcal{D}(A)$ such that $supp(\sigma(\rho)) \subseteq A(last(\rho))$. Intuitively, a policy maps a finite run to a distribution over the set of available actions from the last state of that run. A policy is *stochastic* if $|supp(\sigma(\kappa q))| = |A(q)|$ for every history κq. A run $\rho = q_0a_0q_1\ldots$ is a σ-run if $a_i \in supp(\sigma(q_0a_0\ldots q_i))$. Given a measurable set of runs $P \subseteq \mathrm{Runs}^M$ or finite runs $P \subseteq \mathrm{FRuns}^M$, $\Pr_\sigma[P]$ is the probability that a σ-run belongs to P. We use Runs^{M^σ} to denote the set of all σ-runs and by Π_M the set of all policies over M.

ω-Regular Objectives and (Almost) Sure Winning. Given an MDP M, an *objective* is defined as a set of runs $\Phi \subseteq \mathrm{Runs}^M$. An ω-regular objective can be canonically represented by a *parity objective* (possibly with a larger set of states [7]) $\Phi = \textsc{Parity}[c]$ which is defined using a coloring function $c : Q \to [0;d]$ that assigns each state a *color*. The parity objective $\textsc{Parity}[c]$ contains all runs $\rho \in \mathrm{Runs}^M$ for which the highest color (as assigned by the coloring function c) appearing infinitely often is even. To define them formally, let us write $\mathtt{Inf}_Q(\rho)$ (resp. $\mathtt{Inf}_{Q\times A}(\rho)$) denoting the set of all states (resp. state-action pairs) which occur infinitely often along ρ. Then, the parity objective is defined as $\textsc{Parity}[c] = \{\rho \in \mathrm{Runs}^M \mid \max\{c(q) \mid q \in \mathtt{Inf}_Q(\rho)\}$ is even$\}$, The parity objective $\textsc{Parity}[c]$ reduces to a *Büchi* objective, if the domain of c is restricted to two colors $\{1,2\}$.

Given an MDP M and an objective $\Phi \subseteq \mathrm{Runs}^M$, a run is said to *satisfy* Φ if it belongs to Φ. A policy σ is said to be *surely* (resp. *almost surely*) *winning* from a state q, if in the MDP $M^q = \langle Q, A, \Delta, q\rangle$, every σ-run satisfies Φ (resp. $\Pr_\sigma[\Phi] = 1$). We collect all such states from which a surely (resp. almost surely) winning policy exists in the winning region $\mathcal{W}^\bullet_\Phi$ (resp. $\mathcal{W}^\circ_\Phi$). We say a policy σ is *surely* (resp. *almost surely*) *winning* in MDP M for objective Φ, denoted by $(M,\sigma) \models_\bullet \Phi$ (resp. $(M,\sigma) \models_\circ \Gamma$), if it is surely (resp. almost surely) winning from every state in winning region.

Standard algorithms for computing winning policies for sure (resp. almost sure) parity objectives in MDPs reduce the problem to 2-player (resp. $1\frac{1}{2}$-player) parity games [1]. This is obtained by partitioning the states of the MDP into two sets: the system player controls the states in one set and chooses the next action; the environment/random player controls the states in the other set and chooses the distribution over the next states.

Strategy Templates. *Strategy templates* [5] collect an infinite number of strategies over a (stochastic) game in a concise data structure. With the standard reduction of MDPs with parity objectives to 2-player (or $1\frac{1}{2}$-player) parity games, we can use strategy templates to represent winning policies in MDPs.

Given an MDP $M = \langle Q, A, \Delta, q_0\rangle$ and a strategy template is a tuple $\Gamma = (S, D, H_\ell)$ comprising a set of *unsafe* state-action pairs $S \subseteq Q \times A$, a set of *co-live* state-action pairs $D \subseteq Q \times A$, and a set of *live-groups* $H_\ell \subseteq 2^{Q\times A}$. A strategy template Γ over M induces

the following set Runs^Γ of infinite runs

$$\left\{ \rho \in \text{Runs}^M \;\middle|\; \begin{array}{ll} & \forall (q,a) \in S : qa \notin \rho \\ \wedge & \forall (q,a) \in D : qa \notin \texttt{Inf}_{Q\times A}(\rho) \\ \wedge & \forall H \in H_\ell : src(H) \cap \texttt{Inf}_Q(\rho) \neq \emptyset \\ & \quad \rightarrow H \cap \texttt{Inf}_{Q\times A}(\rho) \neq \emptyset \end{array} \right\}$$

where $src(H) = \{q \mid \exists a : (q,a) \in H\}$. Intuitively, a run $\rho \in \text{Runs}^\Gamma$ satisfies the following conditions: (i) ρ never uses the unsafe actions in S, (ii) ρ stops using the co-live actions in D eventually, and (iii) if ρ visits the set of source states of a live-group $H \in H_\ell$ infinitely often, then it also uses the actions in H infinitely many times. Given an MDP M, a policy σ in M *follows* a template Γ if $(M,\sigma) \models_\bullet \text{Runs}^\Gamma$. If M is clear from the context we often abuse notation and write $\sigma \models_\bullet \Gamma$ if σ follows Γ. A strategy template Γ is said to be *surely* (resp. *almost surely*) winning for the parity objective Φ if every policy that follows Γ is surely (resp. almost surely) winning for Φ.

Existing algorithms [5, 33] compute a *surely* (resp. *almost surely*) *winning* strategy template $\Gamma_\bullet$ (resp. $\Gamma_\circ$) for a (stochastic) parity game. Using these algorithms along with the standard reduction of MDPs to 2-player (or $1\frac{1}{2}$-player) parity games, we obtain a winning strategy template for an MDP. We denote the algorithm that computes a winning strategy template by $\textsc{Template}_\star(M,\Phi)$ with $\star \in \{\bullet,\circ\}$.

3 DYNAMIC ω-REGULAR SHIELDING

This section formalizes our novel dynamic shielding framework via Strategy-Template-based-Adaptive-Runtime-Shields (STARs) schematically depicted in Fig. 2.

3.1 Dynamical Interference via STARs

Given a winning strategy template $\Gamma_\star = (S, D, H_\ell)$ for the parity objective Φ interpret over an MDP, STARs dynamically *blend* a given nominal policy σ with the safety and liveness obligations of Φ localized in $\Gamma_\star$ as depicted in Fig. 3.

Intuitively, to comply with the safety template S, STARs set the probabilities of unsafe actions to zero, preventing runs from reaching states where Φ can no longer be satisfied. For each edge in the co-live group D, STARs maintain a counter for how often the edge is sampled; each sample reduces its probability so that runs eventually avoid co-live edges. Similarly, for each live group H_ℓ, STARs keep a counter for how often the policy visits the group's source states without sampling any of its actions. Each such visit increases the probability of sampling these actions (as a function of the counter), ensuring that if the source states are visited often enough, some action in the live group is eventually sampled with probability close to 1. After an action is sampled, the counter resets and the process repeats. To formalize this intuition, we first define the history-dependent counter functions for co-live and live groups.

Definition 1. *Let M be an MDP and $\Gamma = (S, D, H_\ell)$ a strategy template interpreted over M. Further, let $\kappa = q_0 a_0 q_1 a_1 \dots q_n a_n \in FRuns^M$ be a finite run over M. Then we define for all $(q,a) \in D$: $count_{(q,a)}(\kappa) := |\{i \mid \kappa[i] = (q,a)\}|$, and for all $H \in H_\ell$: $count_H(\kappa) := |\{i > maxpref_H^\kappa \mid \kappa[i] \in \{(q,a) \mid q \in src(H)\}\}|$, where $maxpref_H^\kappa := j$ such that $\kappa[j] \in H$ and $\kappa[j;\infty] \cap H = \emptyset$.*

The above definition lets us formalize how a STAR modifies the distribution $\mu(A(q))$ over actions chosen by σ in the current state q (reached with history κ) using a template $\Gamma_\star$. Intuitively, the resulting counter-based *bias* toward satisfying Φ is *tuned* by an enforcement parameter γ and a threshold θ, both of which can be adjusted dynamically at runtime.

Definition 2 (STARs)**.** *Fix an MDP M, a finite run $\kappa \in FRuns^M$ with $q = last(\kappa)$, a strategy template Γ interpreted over M, an enforcement parameter γ, and a threshold θ. Then, the probability distribution $\mu \in \mathcal{D}(A(q))$ induces a* shielded distribution $\overline{\mu} \in \mathcal{D}(A(q))$ *with $\overline{\mu} := \mathcal{N}(\mu')$ s.t. for all $a \in A(q)$*

$$\mu'(a) := \begin{cases} 0 & \text{if } \mathcal{N}(\mu'')(a) \leq \theta, \\ \mathcal{N}(\mu'')(a) & \text{otherwise.} \end{cases} \tag{1a}$$

$$\mu''(a) := \begin{cases} 0 & \text{if } (q,a) \in S, \\ \mu(a) - \gamma \cdot count_{(q,a)}(\kappa) & \text{if } (q,a) \in D, \\ \mu(a) + \gamma \cdot count_H(\kappa) & \text{if } (q,a) \in H, H \in H_\ell. \end{cases} \tag{1b}$$

We write $\overline{\mu} = STARs(\mu, \kappa, \Gamma, \gamma, \theta)$ to denote that $\overline{\mu}$ is obtained from μ via (1).

The effect of shielding a policy as formalized in Def. 2 is illustrated in Fig. 3. Intuitively, (1b) ensures that the probability of taking certain actions in state q is adapted via the counters induced by the history of the current run and the enforcement parameter γ. For $\gamma \sim 1$, these updates are very aggressive and for $\gamma \sim 0$, they are very mild. As the resulting function μ'' is not a probability distribution anymore (as probabilities over $A(q)$ do not sum up to 1), we normalize it as in (1a) to impose the threshold $\theta > 0$ to make sure that a live edge is surely taken after a finite number of time steps (dependent on γ and σ). In the end, we normalize the resulting distribution to obtain the final distribution.

Remark 1. *We note that if there is an unsafe action a in S such that the current distribution μ assigns a probability of 1 to it, i.e., $\mu(a) = 1$, then $\overline{\mu}$ as in Def. 2 will be not well-defined as it assigns zero probability to all actions. This corner case can be handled by perturbing the distribution μ slightly, e.g., by adding a small $\varepsilon > 0$ to all actions before applying STARs.*

With the definition of shielded distribution in Def. 2, the definition of a *shielded policy* immediately follows.

Definition 3. *Given any MDP M, a strategy template Γ interpreted over M, a threshold θ, and an enforcement parameter $\gamma > 0$, a stochastic policy σ in M induces the shielded policy $\sigma|_\gamma^{\Gamma,\theta} : FRuns^M \mapsto \mathcal{D}(A)$ s.t. $\sigma|_\gamma^{\Gamma,\theta}(\kappa) = STARs(\sigma(\kappa), \kappa, \Gamma, \gamma, \theta)$.*

To avoid the corner case discussed in Rem. 1, the definition assumes that the initial policy σ is stochastic, i.e., $supp(\sigma(\kappa q)) = A(q)$ for all histories κq. This is without loss of generality as any deterministic policy can be converted into a stochastic one as discussed in Rem. 1. Furthermore, as the resulting shielded policy is dependent on the history of a run, a policy is actually shielded via STARs *online* while generating a *shielded run*, as illustrated in Fig. 2 (right). We emphasize that STARs never modify the underlying stochastic policy in place, thereby maximizing modularity between the nominal policy and constraint enforcement.

3.2 Correctness of STARs

Correctness of STARs follows directly from the fact that they are based on *winning strategy templates* $\Gamma_\star$, which implies that the

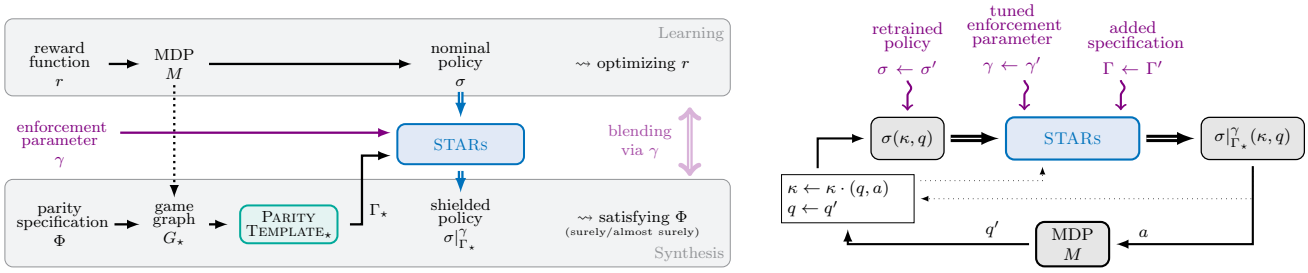


Figure 2: Overview of STARs synthesis (left) and runtime-application of STARs (right). The detailed operation of STARs is illustrated in Fig. 3. Cyan components are taken from the literature. Purple components illustrate dynamic adaptability.

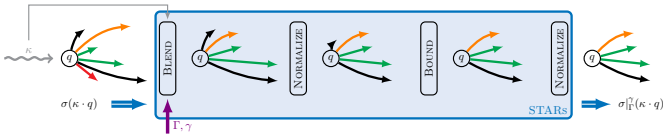


Figure 3: Illustration of dynamic interference via STARs. Length of arrows indicate the relative probability of the corresponding action in $\mu \in \mathcal{D}(A(q))$. The strategy template $\Gamma = (S, D, H_\ell)$ is illustrated via colors red (S), orange (D) and green (H_ℓ). Black arrows are unconstrained. Blending applies (1b) **in Def. 2, bounding applies** (1a) **in Def. 2 and normalizing applies standard normalization, respectively.**

shielded policy $\sigma|_\gamma^{\Gamma_\star,\theta}$ satisfies the objective Φ (almost) surely, if it follows the template. It remains to show that the shielded policy indeed follows the template. As (1b) ensures that the shielded policy assigns zero probability to unsafe edges and that the probability of taking co-live edges is reduced with each visit, the shielded policy will never take an unsafe edge and will eventually avoid co-live edges. Furthermore, as (1a) increments the counter for live groups each time the source states are visited without any action from the group being taken, the shielded policy will eventually take an action from the live group. In total, the shielded policy will follow the template $\Gamma_\star$ and therefore the shielded run satisfies Φ.

Theorem 1. *Given the premises of Def. 3 it holds that $\sigma|_\gamma^{\Gamma_\star,\theta}$ follows $\Gamma_\star$. Moreover, if $\Gamma_\star := \textsc{Template}_\star(M, \Phi)$, then, every $\sigma|_\gamma^{\Gamma_\star,\theta}$-run from the winning region of Φ satisfies Φ surely ($\bullet$) or almost surely ($\circ$).*

3.3 Minimal Interference of STARs

Minimal interference of STARs is, unfortunately, less straightforward to formalize. Based on existing notions, we characterise two orthogonal notions: (i) a minimal deviation in the distribution of observed histories, and (ii) a minimal expected average shielding cost measured in the expected number of non-optimal action choices.

History-based minimal interference is inspired by a similar notion from [16] for safety shields: an action must be deactivated after a history κ, if *and only if* there exists a nonzero probability that the safety constraint would be violated in a bounded extension of κ, regardless of the agent's policy. This argument extends to STARs for both safety and co-live templates, ensuring minimal interference in these settings. However, defining minimal interference for liveness templates is more challenging due to their inherently infinite nature, making bounded violations inapplicable. Instead, we establish minimal interference by showing that for any bounded execution κ that can be extended to a run that satisfies the liveness template, the probability of observing κ in the shielded execution remains close to its probability under the nominal policy.

Theorem 2. *Given the premises of Thm. 1 with $\sigma' = \sigma|_\gamma^{\Gamma_\star,\theta}$, for any $\varepsilon > 0$ and for all $l \in \mathbb{N}$, there exist parameters $\gamma, \theta > 0$ s.t. for all histories κ of length l with $\kappa \in \mathit{pref}(\Phi)$, it holds that $\Pr_{\sigma'}(\kappa) > \Pr_\sigma(\kappa) - \varepsilon$, where $\mathit{pref}(\Phi)$ is the set of prefixes of runs satisfying Φ.*

Minimal shielding costs are inspired by a similar notion in [6], where a cost function is used to measure how much a (deterministic) shield changes the action choices of a (pure) policy. A natural extension of this notion to stochastic policies is to define a shielding cost based on the distance between the distributions of the actions taken by the shielded and the nominal policies. Thereby, the shielding cost can also vary based on the history of the run, allowing for a more fine-grained analysis as in [6]. To formalize this, we define a history-based cost function $\texttt{cost} : \text{FRuns} \to [0, W]$ that assigns a cost to the history κ and the cost of shielding the policy σ at κ is defined as $\texttt{cost}(\kappa, \sigma, \sigma') = \texttt{cost}(\kappa) \cdot \mathrm{D_{TV}}(\sigma(\kappa), \sigma'(\kappa))$. This cost captures the interference as the difference between the action distributions of the original and shielded policies based on the cost of shielding at κ. This can be generalized to the cost of a run ρ as $\texttt{cost}(\rho, \sigma, \sigma') = \limsup_{l\to\infty} \frac{1}{l} \sum_{i=0}^{l-1} \texttt{cost}(\rho[0;i], \sigma, \sigma')$, capturing the average cost of the difference between both policies over a run.

The following theorem states that the expected average cost of the shielded policy is bounded when the template contains only liveness templates. This restriction is needed because satisfying the ω-regular constraint may require the shielded policy to stop (or eventually stop) taking unsafe (or co-live) actions.

Theorem 3. *Given the premises of Thm. 1 with $\sigma' = \sigma|_\gamma^{\Gamma_\star,\theta}$, $\Gamma_\star = (\emptyset, \emptyset, H_\ell)$, and cost function $\texttt{cost} : \mathit{FRuns}^M \to [0, W]$, for any $\varepsilon > 0$, suitable $\gamma, \theta > 0$ exist such that $\mathbb{E}_{\rho\sim\sigma'}\texttt{cost}(\rho, \sigma, \sigma') < \varepsilon$.*

We note that the corner case discussed in Rem. 1 does not compromise these results. In Thm. 2, any history κ satisfying $\kappa \models \mathit{pref}(\Phi)$ inherently avoids unsafe actions. In Thm. 3, the assumption that the strategy template excludes unsafe and co-live actions ensures that the bound on the expected average cost remains valid.

3.4 Dynamic Adaptations of STARs

So far, we have considered a shielding scenario for a *static* parity objective Φ. However, a major strength of strategy templates is their *efficient compositionality* and *fault-tolerance*, which allow for further dynamic adaptations of STARs.

Compositionality facilitates the incremental integration of multiple ω-regular specifications into STARs. By using the existing algorithm ComposeTemplate from [5, Alg.4] we can compute STARs for *generalized parity constraints* of the form $\Phi = \wedge_{i=1}^{k} \Phi_i$, where each Φ_i represents a parity constraint over $G_\star^M$. Crucially, these objectives Φ_i may not be available all at once but might arrive incrementally over time, leading to the need to update the applied shield at runtime. As ComposeTemplate simply combines strategy templates for all objectives into a single (non-conflicting) template, Thm. 1 and Thm. 2 also apply in this case, as long as the run is in the combined winning region of all objectives during the update.

Fault-tolerance ensures that STARs correctly handles occasional or persistent action unavailability. Persistent faults are addressed by marking actions unsafe and resolving conflicts as needed (see [5, Alg.5]), while occasional faults are handled by temporarily excluding unavailable actions from the template (see [5, Sec.5.2]).

Remark 2. *A common assumption in robotic applications is that (incrementally arriving)* liveness *specifications are satisfiable from every safe node in the workspace [14, 18, 24, 31, 40] – most robotic systems can simply invert their path by suitable motions to return to all relevant positions in the workspace. Using the common decomposition of ω-regular objectives φ into a safety part φ_s and a liveness part φ_ℓ, one can restrict the newly arriving objectives to liveness obligations only. In such applications, incremental synthesis never leads to a decreased winning region. This is in fact the case in the incremental instances considered in the FactoryBot benchmark used for evaluation.*

3.5 Optimality of STARs in Rewardful MDPs

While we already formalized *correctness and minimal interference* of STARs, now we strengthen this result further, i.e., we show that whenever nominal policies have been computed to optimize a given reward function (under certain assumptions), STARs produce a shielded policy which achieves a reward which is ϵ-close to the nominal one while additionally guiding the agent to (almost) surely satisfy an ω-regular correctness specification.

An optimal policy over an MDP is typically computed (e.g. via reinforcement learning (RL)) by associating transitions with *reward functions* which are typically Markovian and assign utility to state-action pairs. Formally, a *rewardful MDP* is denoted as a tuple (M, r) where M is an MDP equipped with a reward function $r : Q \times A \mapsto \mathbb{R}$. Such an MDP under a policy σ determines a sequence of random rewards $r(X_i, Y_i)$ for $i \geq 0$, where X_i and Y_i are the random variables denoting the i^{th} state and i^{th} action, respectively. Given a rewardful MDP (M, r) with initial state q_0 and a policy σ, we define the *discounted reward* and the *average reward*

$$\mathrm{Disc}_\sigma^{q_0}(\lambda) := \lim_{N\to\infty} \mathbb{E}_\sigma^{q_0}\left(\sum_{0\leq i\leq N} \lambda^i r(X_i, Y_i)\right)$$
$$\mathrm{Avg}_\sigma^{q_0} := \limsup_{N\to\infty} \tfrac{1}{N}\mathbb{E}_\sigma^{q_0}\left(\sum_{0\leq i\leq N} r(X_i, Y_i)\right).$$

For discounted rewards, the impact of obtained rewards decreases with time. Therefore, the optimal reward achievable by a policy over a given MDP significantly depends on the bounded (initial) executions possible over this MDP. Hence, the ϵ-closeness follows directly from Thm. 2.

For average rewards, the optimal average reward is equivalently impacted by rewards collected over the entire (infinite) length of runs compliant with the policy. This implies, that a policy can only satisfy an ω-regular property (almost) surely *and* optimize the average reward, if it can "switch" between their satisfaction by alternating infinitely often between finite intervals which satisfy either one. This, however, is only possible if the underlying MDP is 'nice' enough to allow for this alternation. In particular, it is known that optimal average rewards is usually obtained by taking the maximum over the achievable rewards in each *good-end components* of the MDP (see [1] for details). Therefore, the ϵ-closeness of the average reward can be guaranteed when the MDP is a good-end component for the specification.

Theorem 4. *Given the premises of Thm. 1 with $\sigma' = \sigma|_\gamma^{\Gamma_\star,\theta}$ and $\mathcal{W}_\Phi^\star = Q$, for every $\varepsilon > 0$, with suitable parameters $\theta, \gamma > 0$ it holds that $Disc_{\sigma'}^{q_0}(\lambda) > Disc_\sigma^{q_0}(\lambda) - \varepsilon$. Furthermore, if the MDP is a good-end component, then with suitable parameters, it holds that $Avg_{\sigma'}^{q_0} > Avg_\sigma^{q_0} - \varepsilon$.*

Remark 3. *We remark that the assumption $\mathcal{W}_\Phi^\star = Q$ in Thm. 4 is not restrictive for two main reasons. First, if $\mathcal{W}_\Phi^\star \subsetneq Q$, we can use $\mathcal{W}_\Phi^\star$ as an additional constraint in existing safe reinforcement learning frameworks. For instance, one can use* preemptive safety shielding *proposed in [2]. This allows to learn an optimal policy within $\mathcal{W}_\Phi^\star$ which directly allows to transfer the results from Thm. 4. Second, we recall the discussion of Rem. 2 to note that including the safety-part of the objective into the learning process does not harm the incremental adaptation of STARs when new (liveness) specifications arrive.*

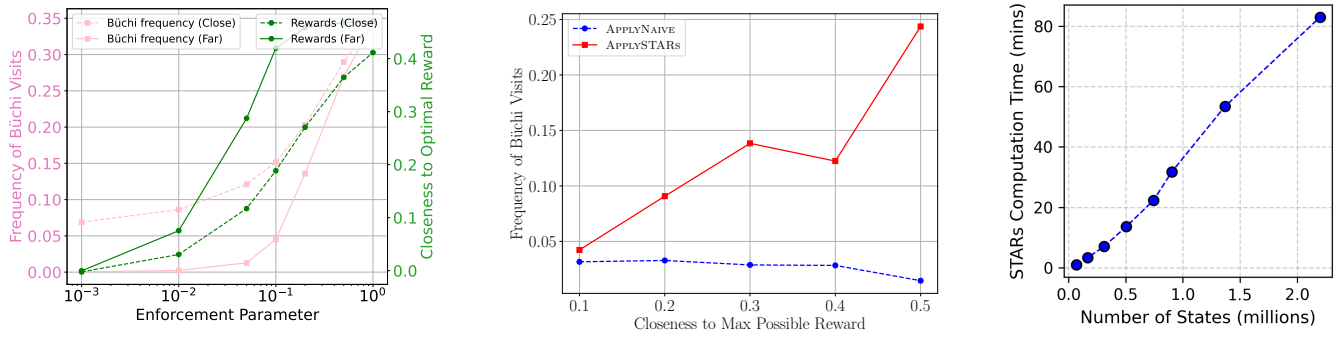
4 EXPERIMENTS

We empirically evaluate STARs on three benchmarks introduced in Sec. 1.1 to demonstrate their effectiveness in: (i) enforcing ω-regular (including liveness) properties at runtime, (ii) enabling dynamic specification updates without retraining, and (iii) scaling to complex environments. To showcase this, we have implemented our shielding algorithm ApplySTARs (as depicted in Fig. 2) in a Python-based prototype tool MARG (Monitoring and Adaptive Runtime Guide) [3]. The experiments were run on a 32-core Debian machine with an Intel Xeon E5-V2 CPU (3.3 GHz) and up to 256 GB of RAM. Recordings of some simulations are available at http://anonymous.4open.science/w/MARGA.

4.1 FactoryBot Benchmarks

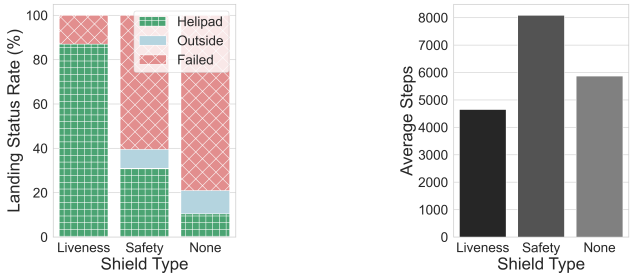
We evaluate our tool on a benchmark suite FactoryBot for guiding a robot in a grid world (see Fig. 1a and 1b). As noted in Sec. 1.1, the robot follows a policy σ trained to maximize average reward on randomly generated grids and then shielded with dynamic safety and liveness specifications (e.g., goal visiting, obstacle avoidance) using our tool. These specifications are programmatically modified to simulate deployment-time constraint changes.

Unfortunately, the closest liveness shielding tool from Avni et al. [6] is not publicly available. Moreover, our evaluation focuses on



(a) (b) (c)

Figure 4: Evaluation summary. (a) Effect of γ on the Büchi frequency and average reward for ApplySTARs in FactoryBot (b) Büchi frequency vs. average reward for ApplySTARs and ApplyNaive (c) Scalability of ApplySTARs on Overcooked-AI.



(a) (b)

Figure 5: (a) Rate of different landing outcomes in LunarLander (b) Average steps to successfully land in LunarLander.

dynamic interference via online tuning of the enforcement parameter γ, a feature not supported by existing methods. We also discuss a comparison against a naïve shielding baseline.

Experimental Setup. We generate random square grids (with sizes from 5 to 13) by sampling walls and reward zones. The reward region is placed at an ℓ_1-distance between min_dist·size and max_dist·size from every cell in the Büchi region ($\mathcal{B}$). We construct 383 instances: 189 `Far` (min_dist = 0.7, max_dist = 0.9) and 194 `Close` (min_dist = 0.1, max_dist = 0.2) instances. We start with a policy that maximizes average reward without the Büchi objective, then ApplySTARs is applied. For each instance, we measure the number of Büchi region visits and the average reward over 100,000 steps from a random initial state.

Tuning γ. We analyze the trade-off between the frequency of visits to the Büchi region (Büchi frequency) and the average reward achieved by ApplySTARs. To achieve higher average rewards, the robot must spend more time in the high payoff region ($\mathcal{R}$). We evaluate this trade-off by systematically varying the enforcement parameter γ in ApplySTARs and, for each value, measure the average Büchi frequency and average reward across all instances that achieve an average reward within ε-close (for $\varepsilon \in \{0.1, \ldots, 0.5\}$) of the maximum possible for that instance. This evaluation is performed separately for the `Far` and `Close` categories. As the robot needs to remain longer in $\mathcal{R}$ to increase the average reward, the distance between $\mathcal{B}$ and $\mathcal{R}$ becomes a key factor: as the distance increases, γ needs to be smaller to attain a given closeness ε to the average reward.

This theoretical dependence is supported by Fig. 4a which shows the Büchi frequency (pink) and the proximity to the maximum average reward (green) attained by ApplySTARs for a given enforcement parameter γ, over instances from `Far` (dashed) and `Close` (solid), respectively. We observe that as the enforcement parameter increases, the Büchi frequency increases while the average reward gets further away from the optimal for both classes of instances. As expected, these trends have a higher slope on `Far` instances.

Remark 4. *We report only on experiments with optimal average reward policies, as discounted rewards pose less challenge for shielding. Discounted reward policies depend on initial trace segments, while ω-regular objectives can be satisfied regardless of any finite prefix (if started in the winning region). This enables a trivial shielding strategy in the FactoryBot benchmark: use a low γ initially, then increase it later. While this yields high performance for both objectives, it is only possible because STARs support dynamic γ updates at runtime—a key advantage over existing methods.*

Quality of Shielded Policies. Given the fact that Thm. 4 restricts attention to *good end components* (i.e. the maximum color in the end component is even), we note that any stochastic policy σ satisfies Φ almost surely within each good end component $\widetilde{Q}$ already without any shielding. This is due to the fact that under stochastic policies all edges have positive probability of being sampled and, therefore, infinite runs reach all states in the SCC almost surely. As the maximum color in $\widetilde{Q}$ is even, all runs satisfy Φ almost- surely. In practice, however, the frequency with which even color vertices are seen is extremely low. While one might suggest that perturbing the nominal policy σ might increase the frequency of visiting even color states, this is actually not the case, as this perturbed policy would explore the *entire state space* more aggressively. We call the algorithm that implements this perturbation ApplyNaive.

In contrast, STARs modify probabilities in a targeted fashion. This (i) avoids visiting odd-color vertices which are not optimal, and (ii) allows to tune the desired frequency in which even color vertices are visited via the enforcement parameter. This can be formalized using the notion of *frequency* of a run ρ visiting a set T of states which

can be defined as $\texttt{freq}(\rho, T) = \limsup_{l\to\infty} \frac{1}{l} |\{i \in [0; l] \mid \rho[i] \in T\}|$. With this definition, the following theorem[4] ensures that the frequency of a run ρ visiting even color states can be increased by tuning the enforcement parameter γ.

Theorem 5. *Given the premises of Thm. 4 with $\sigma' = \sigma|_\gamma^{\Gamma_\bullet,\theta}$ and T being the set of even color states in the objective Φ, for every frequency $0 < \delta \leq \frac{1}{|Q|}$, there exists parameters $\theta, \gamma > 0$ such that for every run $\rho \sim \sigma'$, it holds that $\texttt{freq}(\rho, T) \geq \delta$.*

This result is supported by Fig. 4b, which compares our shielding approach ApplySTARs with the naive approach ApplyNaive on the FactoryBot benchmark suite. Fig. 4b plots the average Büchi frequency (y-axis) for all instances that obtained an average reward that is ε-close to the maximal possible reward in that instance, where $\varepsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ (x-axis). The red line shows averages for the robot shielded by ApplySTARs, and the dashed blue line for ApplyNaive. We see that ApplySTARs achieves a similar average reward while ensuring a much higher Büchi frequency than ApplyNaive. Moreover, ApplyNaive cannot increase Büchi frequency beyond a very low level, whereas ApplySTARs can attain very high Büchi frequency by trading off average reward.

4.2 Overcooked-AI Benchmarks

We assess the scalability of our shield computation via the Overcooked-AI environment [11], a widely used benchmark for cooperative multi-agent reinforcement learning. Here, autonomous agents are trained to repeatedly perform cooking tasks. We use LTL specifications to encode additional recipe requirements of produced dishes. STARs are used to enforce the (additional) production of soups satisfying these requirements infinitely often.

From a shielding perspective, Overcooked-AI is very similar to FactoryBot as it is based on a known finite MDP and the LTL recipe objective gets translated into a Büchi objective on the two-player game graph extracted from the known MDP. Therefore, the insights about tuning the enforcement parameter and improving shielding quality discussed in Sec. 4.1 identically apply to Overcooked-AI. Here 'far' and 'close' instances (cf. Fig. 4a) however relate recipes rather than grid states. If additional recipe constraints align with the recipe the agent policy was learned on no shielding is needed and both types of dishes are produced with a similar frequency. If recipes are complementary, γ can be used to balance the frequency of produced dishes.

The main difference between FactoryBot and Overcooked-AI is the size of the naturally arising game arenas which is why we choose Overcooked-AI benchmarks to demonstrate the scalability of ApplySTARs. Towards this goal, we utilize with the `cramped_room` layout with increasing sizes. Within this layout, two agents can move in four directions and perform actions such as picking up ingredients, serving soups, and placing items on counters. We construct the game graph naively by enumerating all possible states and actions, resulting in graphs ranging from 68000 to 2.2 million states across different layout sizes. We compute STARs for each instance, with an overview of state counts and computation times presented in Fig. 4c. These results demonstrate that the synthesis of STARs scales to million-state environments with practical one-time computation time cost (about 1 hour) prior to deployment. We suspect that improved game graph extractions that cluster states with similar shielding requirements can further improve the scalability of ApplySTARs and is an interesting direction for future work.

4.3 LunarLander Benchmarks

We finally evaluate ApplySTARs on the LunarLander benchmark [9]. The standard environment is modified to have more height and to have a randomly positioned helipad. We also modify the reward to not depend on landing on the pad[5].

Experimental Setup. We train a baseline policy using standard proximal policy optimization (PPO) for 50000 time steps, which ensures that the lander touches down safely—though not necessarily on the helipad. Then, to apply STARs, we introduce a 60×60 grid which discretizes only the x/y coordinates of the hidden 8-dimensional MDP of the lunar lander. We further reinterpret the lander actions to move along those grid cells, which vastly simplifies the actual dynamics of the lunar lander. We extract a game from this grid equipped with a safety (preventing the lander from leaving the environment) and a liveness (steering toward the helipad) objective and use it to synthesize STARs. We then ApplySTARs on the trained policy. For comparison, we also evaluate a safety shield which solely prevents the lander from leaving the environment or landing outside the helipad region.

Results. We randomly generated 200 seeds to initiate the environment, and compare the result of policies being (i) unshielded, (ii) shielded with safety shield, and (iii) shielded with STARs. The results are summarized in Fig. 5. The results show that the unshielded policy lands on the helipad in 10.5% cases, while the safety shielded policy lands on the helipad in 31% cases, and our shielded policy lands on the helipad in 87% cases. Also, the average steps to land on the helipad are 5868 for unshielded, 8082 for safety shielded, and 4650 for liveness shielded policies showing that liveness steers the lander toward the helipad more quickly. This showcases the effectiveness of STARs in ensuring liveness while preserving the learnt policy to not crash the lander.

In particular, this behavior is observed even though the actual dynamics of the lander are very complex, leading to quite different abstract behavior as captured by our small game abstraction used to compute STARs. This showcases the applicability of STARs to complex high-dimensional environments. Combined with the scalability results on Overcooked-AI we believe that a future scaling of STARs towards industrial scale benchmarks is possible.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback and suggestions. This work was supported by DFG project 89792660 as part of TRR248-CPEC, and by the Emmy Noether Grant SCHM 3541/1-1. We also thank Stratis Tsirtsis for valuable discussions on the paper.

[4]Note that this shows the existence of such parameters only for the case of surely satisfying Φ. For the case of almost-sure satisfaction, the frequency would also depend on the transition probabilities of the MDP and hence, we cannot guarantee the existence of such parameters for every δ, while the same intuition still holds.

[5]The details of the exact modifications to the standard environment are described in the full version of the paper [4].

REFERENCES

[1] Shaull Almagor, Orna Kupferman, and Yaron Velner. 2016. Minimizing Expected Cost Under Hard Boolean Constraints, with Applications to Quantitative Synthesis. In *CONCUR (LIPIcs, Vol. 59)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:15.

[2] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[3] Ashwani Anand. 2025. MARG (Monitoring and Adaptive Runtime Guide). https://github.com/ashwanianand/marg

[4] Ashwani Anand, Satya Prakash Nayak, Ritam Raha, and Anne-Kathrin Schmuck. 2025. Follow the STARs: Dynamic ω-Regular Shielding of Learned Policies. arXiv:2505.14689 [cs.AI] https://arxiv.org/abs/2505.14689

[5] Ashwani Anand, Satya Prakash Nayak, and Anne-Kathrin Schmuck. 2023. Synthesizing Permissive Winning Strategy Templates for Parity Games. In *CAV (1) (Lecture Notes in Computer Science, Vol. 13964)*. Springer, 436–458.

[6] Guy Avni, Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, Bettina Könighofer, and Stefan Pranger. 2019. Run-Time Optimization for Learned Controllers Through Quantitative Games. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, 630–649. https://doi.org/10.1007/978-3-030-25540-4_36

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

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[10] Damien Busatto-Gaston, Debraj Chakraborty, Shibashis Guha, Guillermo A. Pérez, and Jean-François Raskin. 2021. Safe Learning for Near-Optimal Scheduling. In *QEST (Lecture Notes in Computer Science, Vol. 12846)*. Springer, 235–254.

[11] Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca Dragan. 2020. On the Utility of Learning about Humans for Human-AI Coordination. arXiv:1910.05789 [cs.LG] https://arxiv.org/abs/1910.05789

[12] Krishnendu Chatterjee and Laurent Doyen. 2010. Energy Parity Games. In *ICALP (2) (Lecture Notes in Computer Science, Vol. 6199)*. Springer, 599–610.

[13] Krishnendu Chatterjee and Laurent Doyen. 2011. Games and Markov Decision Processes with Mean-Payoff Parity and Energy Parity Objectives. In *MEMICS (Lecture Notes in Computer Science, Vol. 7119)*. Springer, 37–46.

[14] Ziyang Chen, Mingyu Cai, Zhangli Zhou, Lin Li, and Zhen Kan. 2024. Fast Motion Planning in Dynamic Environments With Extended Predicate-Based Temporal Logic. *IEEE Transactions on Automation Science and Engineering* (2024), 1–15. https://doi.org/10.1109/TASE.2024.3418409

[15] David Dalrymple, Joar Skalse, Yoshua Bengio, Stuart Russell, Max Tegmark, Sanjit Seshia, Steve Omohundro, Christian Szegedy, Ben Goldhaber, Nora Ammann, et al. 2024. Towards Guaranteed Safe AI: A Framework for Ensuring Robust and Reliable AI Systems. *arXiv preprint arXiv:2405.06624* (2024).

[16] Ingy Elsayed-Aly, Suda Bharadwaj, Christopher Amato, Rüdiger Ehlers, Ufuk Topcu, and Lu Feng. 2021. Safe Multi-Agent Reinforcement Learning via Shielding. In *AAMAS*. ACM, 483–491.

[17] Let's Build The Future. 2025. Amazon's Warehouse Robots Got Stuck—A Hilarious AI Glitch. https://www.youtube.com/watch?v=AAhHmquT9Fs

[18] David Gundana and Hadas Kress-Gazit. 2021. Event-Based Signal Temporal Logic Synthesis for Single and Multi-Robot Tasks. *IEEE Robotics and Automation Letters* 6, 2 (2021), 3687–3694. https://doi.org/10.1109/LRA.2021.3064220

[19] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. 2023. Multi-objective ω-Regular Reinforcement Learning. *Form. Asp. Comput.* 35, 2, Article 12 (July 2023), 24 pages. https://doi.org/10.1145/3605950

[20] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. 2023. Omega-Regular Reward Machines. In *ECAI (Frontiers in Artificial Intelligence and Applications, Vol. 372)*. IOS Press, 972–979.

[21] Klaus Havelund and Grigore Roşu. 2002. Synthesizing monitors for safety properties. In *Tools and Algorithms for the Construction and Analysis of Systems: 8th International Conference, TACAS 2002 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8–12, 2002 Proceedings 8*. Springer, 342–356.

[22] Kai-Chieh Hsu, Haimin Hu, and Jaime F. Fisac. 2024. The Safety Filter: A Unified View of Safety-Critical Control in Autonomous Systems. *Annual Review of Control, Robotics, and Autonomous Systems* 7, Volume 7, 2024 (2024), 47–72. https://doi.org/10.1146/annurev-control-071723-102940

[23] Loïc Hélouët, Nicolas Markey, and Ritam Raha. 2019. Reachability Games with Relaxed Energy Constraints. In *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019, Bordeaux, France, 2-3rd September 2019 (EPTCS, Vol. 305)*, Jérôme Leroux and Jean-François Raskin (Eds.). 17–33. https://doi.org/10.4204/EPTCS.305.2

[24] Samarth Kalluraya, George J. Pappas, and Yiannis Kantaros. 2023. Multi-Robot Mission Planning in Dynamic Semantic Environments. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 1630–1637. https://doi.org/10.1109/ICRA48891.2023.10160344

[25] Milad Kazemi, Mateo Perez, Fabio Somenzi, Sadegh Soudjani, Ashutosh Trivedi, and Alvaro Velasquez. 2022. Translating Omega-Regular Specifications to Average Objectives for Model-Free Reinforcement Learning. In *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 732–741.

[26] Bettina Könighofer, Roderick Bloem, Rüdiger Ehlers, and Christian Pek. 2022. Correct-by-Construction Runtime Enforcement in AI–A Survey. In *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Springer, 650–663.

[27] Bettina Könighofer, Julian Rudolf, Alexander Palmisano, Martin Tappler, and Roderick Bloem. 2023. Online shielding for reinforcement learning. *Innovations in Systems and Software Engineering* 19, 4 (2023), 379–394.

[28] Jan Kretínský, Guillermo A. Pérez, and Jean-François Raskin. 2018. Learning-Based Mean-Payoff Optimization in an Unknown MDP under Omega-Regular Constraints. In *CONCUR (LIPIcs, Vol. 118)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:18.

[29] Insup Lee, Sampath Kannan, Moonzoo Kim, Oleg Sokolsky, and Mahesh Viswanathan. 1999. Runtime assurance based on formal specifications. In *International Conference on Parallel and Distributed Processing Techniques and Applications*. PDPTA', 279–287.

[30] Haritz Odriozola-Olalde, Maider Zamalloa, and Nestor Arana-Arexolaleiba. 2023. Shielded Reinforcement Learning: A review of reactive methods for safe learning. In *SII*. IEEE, 1–8.

[31] Adam Pacheck and Hadas Kress-Gazit. 2023. Physically Feasible Repair of Reactive, Linear Temporal Logic-Based, High-Level Tasks. *IEEE Transactions on Robotics* 39, 6 (2023), 4653–4670. https://doi.org/10.1109/TRO.2023.3304009

[32] Shashank Pathak, Luca Pulina, and Armando Tacchella. 2018. Verification and repair of control policies for safe reinforcement learning. *Applied Intelligence* 48, 4 (April 2018), 886–908. https://doi.org/10.1007/s10489-017-0999-8

[33] Kittiphon Phalakarn, Sasinee Pruekprasert, and Ichiro Hasuo. 2025. Winning Strategy Templates for Stochastic Parity Games Towards Permissive and Resilient Control. In *Theoretical Aspects of Computing – ICTAC 2024*, Chutiporn Anutariya and Marcello M. Bonsangue (Eds.). Springer Nature Switzerland.

[34] Peter JG Ramadge and Walter Murray Wonham. 1989. The control of discrete event systems. *Proc. IEEE* 77, 1 (1989), 81–98.

[35] Robert Reed, Hanspeter Schaub, and Morteza Lahijanian. 2024. Shielded Deep Reinforcement Learning for Complex Spacecraft Tasking. *arXiv preprint arXiv:2403.05693* (2024).

[36] Yagiz Savas, Christos K. Verginis, Michael Hibbard, and Ufuk Topcu. 2024. On Minimizing Total Discounted Cost in MDPs Subject to Reachability Constraints. *IEEE Trans. Automat. Control* 69, 9 (2024), 6466–6473. https://doi.org/10.1109/TAC.2024.3384834

[37] Sanjit A Seshia, Dorsa Sadigh, and S Shankar Sastry. 2022. Toward verified artificial intelligence. *Commun. ACM* 65, 7 (2022), 46–55.

[38] Martin Tappler, Andrea Pferscher, Bernhard K. Aichernig, and Bettina Konighofer. 2024. Learning and Repair of Deep Reinforcement Learning Policies from Fuzz-Testing Data . In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 38–50. https://doi.org/10.1145/3597503.3623311

[39] Sanne van Waveren, Christian Pek, Jana Tumova, and Iolanda Leite. 2022. Correct Me If I'm Wrong: Using Non-Experts to Repair Reinforcement Learning Policies. In *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 493–501. https://doi.org/10.1109/HRI53351.2022.9889604

[40] Vasileios Vasilopoulos, Yiannis Kantaros, George J. Pappas, and Daniel E. Koditschek. 2021. Reactive Planning for Mobile Manipulation Tasks in Unexplored Semantic Environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 6385–6392. https://doi.org/10.1109/ICRA48506.2021.9561958

[41] Kim P. Wabersich, Andrew J. Taylor, Jason J. Choi, Koushil Sreenath, Claire J. Tomlin, Aaron D. Ames, and Melanie N. Zeilinger. 2023. Data-Driven Safety Filters: Hamilton-Jacobi Reachability, Control Barrier Functions, and Predictive Methods for Uncertain Systems. *IEEE Control Systems Magazine* 43, 5 (2023), 137–177. https://doi.org/10.1109/MCS.2023.3291885

[42] Wen-Chi Yang, Giuseppe Marra, Gavin Rens, and Luc De Raedt. 2023. Safe Reinforcement Learning via Probabilistic Logic Shields. In *IJCAI*. ijcai.org, 5739–5749.

[43] Christopher K. Zeitler, Kristina Miller, Sayan Mitra, John Schierman, and Mahesh Viswanathan. 2024. Optimizing Rewards while meeting ω-regular Constraints. *RLJ* 5 (2024), 2492–2514.

[44] Weichao Zhou, Ruihan Gao, BaekGyu Kim, Eunsuk Kang, and Wenchao Li. 2020. Runtime-Safety-Guided Policy Repair. *CoRR* abs/2008.07667 (2020). arXiv:2008.07667 https://arxiv.org/abs/2008.07667