Directed Exploration in Reinforcement Learning from Linear Temporal Logic

Marco Bagatella
Department of Computer Science
ETH Zürich, Zürich, Switzerland
mbagatella@ethz.ch

Andreas Krause
Department of Computer Science
ETH Zürich, Zürich, Switzerland

Georg Martius
Max Planck Institute for Intelligent Systems
Tübigen, Germany

Abstract

Linear temporal logic (LTL) is a powerful language for task specification in reinforcement learning, as it allows describing objectives beyond the expressivity of conventional discounted return formulations. Nonetheless, recent works have shown that LTL formulas can be translated into a variable rewarding and discounting scheme, whose optimization produces a policy maximizing a lower bound on the probability of formula satisfaction. However, the synthesized reward signal remains fundamentally sparse, making exploration challenging. We aim to overcome this limitation, which can prevent current algorithms from scaling beyond low-dimensional, short-horizon problems. We show how better exploration can be achieved by further leveraging the LTL specification and casting its corresponding Limit Deterministic Büchi Automaton (LDBA) as a Markov reward process, thus enabling a form of high-level value estimation. By taking a Bayesian perspective over LDBA dynamics and proposing a suitable prior distribution, we show that the values estimated through this procedure can be treated as a shaping potential and mapped to informative intrinsic rewards. Empirically, we demonstrate applications of our method from tabular settings to high-dimensional continuous systems, which have so far represented a significant challenge for LTL-based reinforcement learning algorithms.

1 Introduction

Most reinforcement learning (RL) research has traditionally focused on a standard setting, prescribing the maximization of cumulative rewards in a Markov Decision Process [30, 37]. While this simple formalism captures a variety of behaviors [36], its expressiveness remains limited [1]. In pursuit of a more natural and effective way to specify desired behavior, several works have turned towards logic languages [6, 8, 15, 20, 26]. Originally designed to describe possible paths of a system (with direct applications, e.g., in model checking [2]), Linear Temporal Logic (LTL) [29] has been found to strike an interesting balance between expressiveness and tractability.

Several works [3, 16, 42] have proposed a reward and discounting scheme to distill a policy through RL from an LTL specification. Crucially, this policy optimizes a lower bound on the probability of satisfying the specification [42]. However, this scheme results in a *sparse* reward signal and a potentially *flat* value landscape, thus making exploration a fundamental challenge.

17th European Workshop on Reinforcement Learning (EWRL 2024).

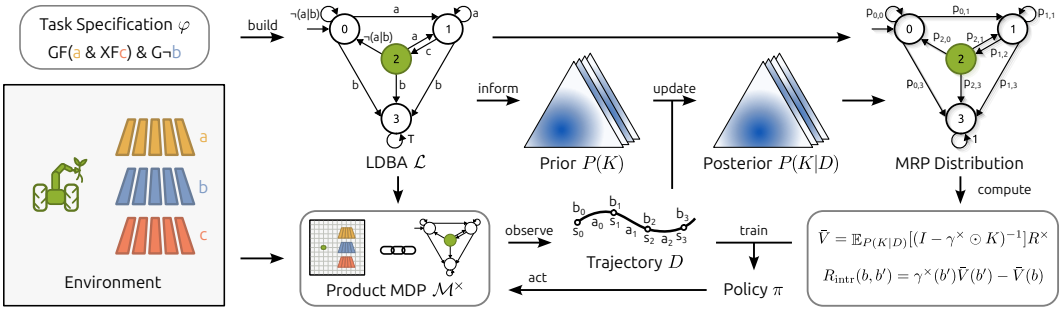

Figure 1: Overview of DRL^2. DRL^2 leverages an LDBA representation of the task (top left) and a Bayesian estimate of its transition kernel (top center) to define a distribution over Markov reward processes (top right), which can be used for high-level value estimation (bottom right). Resulting values are mapped to an intrinsic reward signal (also bottom right), which guides exploration in the product MDP (bottom left).

The necessity for strong exploration algorithms when learning from LTL specification is therefore evident. Existing methods rely on counterfactual data augmentation schemes [42], which however do not directly guide the agent in the underlying MDP, on the availability of a task distribution [44], or on heuristics [16]. Closer in spirit to our work, task-aware reward shaping has briefly been explored in the context of logic on finite traces [6, 20], but has not been scaled to ω-regular problems and cannot adapt to unknown environment dynamics.

The main idea of our work is the distillation of an intrinsic reward signal from the structure of the LTL specification itself. In particular, we repurpose the Limit Deterministic Büchi Automata (LDBA) constructed from an LTL formula as a Markov reward process, by assuming a transition kernel over LDBA states. We then leverage the reward process to perform a form of high-level value estimation and compute values for given LDBA states. These values can be naturally leveraged for potential-based reward shaping. Crucially, we adopt a Bayesian perspective on estimating the transition kernel over the LDBA: by choosing a suitable prior distribution, we ensure that intrinsic rewards are informative from the initial phases of learning. This is done by *optimistically* assuming that the agent is capable of transitioning to any adjacent LDBA state in a single step, although this might not be easily afforded by the dynamics of the environment. Moreover, by updating the distribution according to evidence, the assumed transition kernel can be adapted over time to realistically represent the agent's behavior and environment's dynamics.

Our method, named DRL^2 (**D**irected **R**einforcement **L**earning from Linear Temporal **L**ogic), is illustrated in Figure 1 and is capable of driving deep exploration in reinforcement learning from linear temporal logic specifications. The contributions of this work can be outlined as follows:

1. we design and introduce a method for exploration in reinforcement learning when learning from LTL specifications, by casting the LDBA as a Markov reward process and leveraging it for value estimation and distillation of intrinsic rewards;
2. we analyse the proposed method and the degree of suboptimality potentially induced by intrinsic rewards;
3. we evaluate the method across diverse settings, spanning from simple tabular cases to complex, high-dimensional environments, which pose a significant challenge for RL from LTL specifications.

Section 2 provides an introduction to LTL and its connection to RL. Our method is described in Section 3 and evaluated in Section 4. A discussion of related works and of the proposed one can be found in Sections 5 and 6, respectively.

2 Background

This section provides a brief introduction to linear temporal logic and discusses connections to reinforcement learning. For a complete introduction, we refer the reader to Baier and Katoen [2].

Linear Temporal Logic LTL formulas build upon a finite set of propositional variables (atomic propositions, AP), over which an alphabet is defined as the powerset $\Sigma = 2^{\text{AP}}$, i.e. the combinations of variables evaluating to `true`.

As a more concrete, illustrative example, let us consider a farming robot, which is tasked with continually weeding through any of three different fields. The presence of the robot in each field could be described by the set of atomic propositions $\{a, b, c\}$. When the robot is operating in the first field, a would evaluate to `true` ($\top$), while b and c would evaluate to `false` ($\bot$).

Definition 2.1. (LTL Formula) An LTL formula is a composition of atomic propositions (AP), logical operators `not` ($\neg$), `and` ($\wedge$) and `or` ($\vee$) and temporal operators `next` (X) and `until` (U). Inductively:

- if $p \in AP$, then p is an LTL formula;
- if ϕ and θ are LTL formulas, then $\neg\phi$, $\phi \wedge \theta$, $\phi \vee \theta$, $X\,\phi$ and $\phi\,U\,\theta$ are LTL formulas.

Intuitively, while logical operators encode their conventional meaning, the `next` operator evaluates to `true` if its argument holds true at the very next time step, and `until` is a binary operator which requires its second argument to eventually evaluate to `true`, and its first argument to hold true until this happens. From this sufficient set of operators, additional ones are often defined in order to allow more concise specifications. In the context of reinforcement learning for control, useful operators are `finally` ($F(\phi) := \top U \phi$) and `globally` ($G(\varphi) := \neg F \neg \phi$). Returning to our example, they could be used to specify stability (FGa, i.e., reach and remain in the first field), or avoidance ($G\neg a$, i.e., never enter the first field). A more complex task, which requires the farming robot to visit the first and third fields, repeatedly, while always avoiding the second one, could be simply represented as $(GF(a \wedge XFc)) \wedge (G\neg b)$. Through this work, we will refer to this task as T_0.

Each LTL formula can be *satisfied* by an infinite sequence of truth evaluations of AP (i.e., an ω-*word*). While a direct definition is also possible [38], for simplicity, satisfaction will be introduced through the acceptance of paths in an automaton built from the formula.

From Formulas to Automata A practical way of handling formula satisfaction involves the introduction of Limit Deterministic Büchi Automata (LDBAs). An LDBA can be constructed from any LTL formula [35] and is able to keep track of the progression of its satisfaction.

Definition 2.2. (Limit Deterministic Büchi Automaton – LDBA) An LDBA is a tuple $\mathcal{L} = (\mathcal{B}, \Sigma \cup \mathcal{A}^{\mathcal{B}}, P^{\mathcal{B}}, \mathcal{B}^{\star}, b_0)$, where $\mathcal{B}$ is a finite set of states, $\Sigma = 2^{AP}$ is an alphabet over atomic propositions, $\mathcal{A}^{\mathcal{B}} : \mathcal{B} \to \{0, \dots, N-1\}$ is the subset of indexed jump transitions available at each state, $P^{\mathcal{B}} : \mathcal{B} \times (\Sigma \cup \mathcal{A}^{\mathcal{B}}) \to \mathcal{B}$ is a transition function, $\mathcal{B}^{\star} \subseteq \mathcal{B}$ is a set of accepting states and $b_0 \in \mathcal{B}$ is the initial state.

We remark that jump transitions $\mathcal{A}^{\mathcal{B}}$ are relevant for a subset of formulas including, e.g. stability problems. While our method can be applied independently from their presence, for the sake of simplicity, an extension to jump transitions is presented in Appendix I.

An infinite sequence of LDBA *actions* $(a_i)_0^{\infty} \in (\Sigma)^{\infty}$ induces a *path* $p = (b_i)_0^{\infty}$ according to $b_{i+1} = P^{\mathcal{B}}(b_i, a_i)$.

Definition 2.3. (Acceptance) An LDBA $\mathcal{L}$ accepts a path $(b_i)_0^{\infty}$ if and only if the path visits an accepting state $b^{\star} \in \mathcal{B}^{\star}$ infinitely often, that is $\forall t \in \mathbb{N}, \exists t' > t$ such that $b_{t'} \in \mathcal{B}^{\star}$.

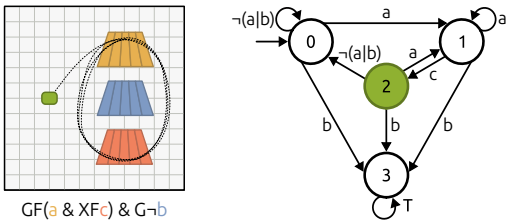

Figure 2: Illustrative task T_0 (left), and LDBA encoding the formula (right, with starting state marked as 0 and accepting state in green). The farming robot (in green) moves in a 2D plane, where three areas of different colors represent fields. $\{a, b, c\}$ are atomic propositions that evaluate to `true` when the agent enters the yellow, blue and red field, respectively. A trajectory satisfying the specification is shown.

By translating each LTL specification into an LDBA, it is now possible to tie formula satisfaction to the acceptance of a path: informally, an infinite sequence of AP evaluations (ω-*word*) satisfies a formula φ if and only if the path $(b_i)_0^{\infty}$ induced by the ω-word in the LDBA $\mathcal{L}$ synthesized from φ is accepted.

Let us consider the example in Figure 2 for the illustrative task T_0 and the LTL formula $\varphi = (GF(a \wedge XFc)) \wedge (G\neg b)$. The periodic LDBA path $(0, 1, 2)^{\infty}$ is accepted, just as $(0, 1, 1, 2)^{\infty}$, although the latter takes on average more steps to reach the accepting state. On the other hand, the

paths $(0)^\infty$ or $(0,(3)^\infty)$ would not be accepted, as neither ever reach the accepting state, with the second getting caught in a sink state by violating the avoidance criterion in φ.

We note that this notion of success relies on conditions that are achieved *eventually* in the future, independently of temporal distance. While this leads to myopic behavior under naive optimization [42], recent works [3, 7, 16, 42] propose an elegant rephrasing for formula satisfaction in an RL-friendly form, as we now describe.

Product MDPs and Policy Optimization The following part describes standard RL terminology and then reconciles it with the introduced logic machinery.

Definition 2.4. (Markov Decision Process – MDP) A Markov Decision Process is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu_0)$, where $\mathcal{S}$ and $\mathcal{A}$ are potentially continuous state and action spaces, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a probabilistic transition kernel[1], $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, γ is a discount factor, and $\mu_0 \in \Delta(\mathcal{S})$ is the initial state distribution.

MDPs are the standard modeling choice for RL environments; however, they are disconnected from atomic propositions and unable to track progression over formula satisfaction. The semantics of APs can be grounded in MDP states through a labeling function $\mathcal{F} : \mathcal{S} \to \Sigma$, which evaluates APs in each MDP state. On the other hand, progression over task satisfaction can naturally be stored in an LDBA, as its states encode sufficient information on the history of paths.

Finally, all three components (MDP, labeling function and LDBA) can be *synchronized* to ensure consistency between trajectories in each of them and enable mapping policies to distribution over paths, and therefore likelihoods of formula satisfaction [15, 42]. This can be done by defining a product MDP $\mathcal{M}^\times = (\mathcal{S}^\times, \mathcal{A}^\times, \mathcal{P}^\times, R^\times, \gamma^\times, \mu_0^\times)$, where $\mathcal{S}^\times = \mathcal{S} \times \mathcal{B}, \mathcal{A}^\times = \mathcal{A}$ and $\mu_0^\times(s,b) = \mu_0(s) \cdot \mathbf{1}_{b=b_0}$.

The transition kernel over $\mathcal{M}^\times$ needs to guarantee that both the underlying MDP and the LDBA evolve consistently. This synchronization is achieved through the labeling function $\mathcal{F}$:

$$P^\times((s',b') \mid (s,b),a) = P(s' \mid s,a), \text{ with } b' = P^{\mathcal{B}}(b, \mathcal{F}(s')). \tag{1}$$

Through this construction, it is finally possible to connect trajectories (and, therefore, policies) to satisfaction of a given LTL formula. Let us consider an LTL formula φ and its corresponding LDBA $\mathcal{L}$, as well as a trajectory $\tau = (s_i, b_i)_0^\infty$ in the product MDP $\mathcal{M}^\times$. Then, $\tau \models \varphi$ (τ satisfies φ) if and only if $\mathcal{L}$ accepts the path $(b_i)_0^\infty$, i.e., the projection of τ to LDBA states. Finally, let us consider a policy $\pi : \mathcal{S}^\times \to \Delta(\mathcal{A}^\times)$: the probability of π satisfying φ can thus be defined as the probability integral for trajectories satisfying the formula: $P(\pi \models \varphi) = \mathbb{E}_{\tau\sim\pi}\, \mathbf{1}_{\tau\models\varphi}$. Optimizing a policy π for satisfaction of an LTL specification φ can be expressed as finding $\pi^\star \in \operatorname{argmax}_{\pi\in\Pi} P(\pi \models \varphi)$. Prior works [15, 42] have proposed RL-friendly proxy objectives, which optimize a lower bound on the probability of LTL formula satisfaction:

$$\pi_\gamma^\star \in \operatorname*{argmax}_{\pi\in\Pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[\sum_{i=0}^{\infty} \Gamma_i R^\times(b_i)\Big]\ (:= V_\pi^\gamma), \text{ where} \tag{2}$$

$$R^\times(b_i) = \mathbf{1}_{\{b_i\in\mathcal{B}^*\}},\ \Gamma_0 = 1,\ \Gamma_i = \prod_{t=0}^{i-1} \gamma^\times(b_t), \quad \gamma^\times(b_t) = \begin{cases} \gamma, & b_t \in \mathcal{B}^* \\ 1, & \text{otherwise.} \end{cases} \tag{3}$$

Paraphrasing, $\pi_\gamma^\star$ maximizes the visitation count to accepting states in the LDBA under *eventual discounting*. For a formal analysis of the policy recovered by this objective, we refer the reader to Voloshin et al. [42]. Our work builds upon this formulation and devises an exploration method to compensate for its drawbacks. That is, the reward function $R^\times$ is fundamentally sparse: the agent only receives feedback when a significant amount of progress toward solving the task has been made, and thus an accepting LDBA state is visited. As a result, while naive exploration might reach several non-accepting LDBA states, the agent remains unable to evaluate them, as it is largely uninformed of the yet unexplored parts of the LDBA. Our method distills the global known structure of the LDBA in a denser intrinsic reward signal for exploration.

[1] $\Delta(\mathcal{S})$ represents the space of probability distributions over $\mathcal{S}$.

3 Method: Directed Exploration in Reinforcement Learning from LTL

Our method relies on (i) repurposing the LDBA as a Markov reward process by assigning a transition kernel, as well as rewards and discount signals for each transition, (ii) defining a value estimation operator to compute high-level values for each LDBA state, and finally (iii) leveraging these values for potential-based reward shaping. This procedure is described in Section 3.1. It takes an LDBA and a transition kernel as input and returns intrinsic rewards for each transition in the product MDP. A second and crucial component, discussed in Section 3.2, is the estimation of the LDBA transition kernel, which is essential for ensuring informative intrinsic rewards: by taking a Bayesian perspective, we show that a symmetric prior can induce strong exploration. Finally, Section 3.3 connects each component and describes a practical instantiation of the algorithm.

3.1 High-level Value Estimation from LDBA

As described in Section 2, an LDBA can be naturally synthesized from a given LTL specification, through well-known schemes [35]. We now show how an LDBA can be recast as a Markov reward process, which can be leveraged for computing values for each LDBA state. This construction requires 2 ingredients, namely (i) an LDBA $\mathcal{L}$, and (ii) a transition kernel K over LDBA states, where K is a stochastic matrix such that $K_{i,j} = \mathbb{E}_{\pi, P^\times, \mu_0^\times} P(b' = b_j | b = b_i)$, i.e., an estimate of the probability for the LDBA to transition to state b_j starting from b_i under some policy π, assuming stationarity. The choice of the transition kernel K is crucial for the effectiveness of the method and is thus treated in detail in the following section.

Having access to these two ingredients, we can define a discrete Markov reward process $\bar{\mathcal{L}} = (\mathcal{B}, K, R^\times, \gamma^\times, b_0)$: the state space $\mathcal{B}$ and initial state b_0 are left unchanged and coupled with the transition kernel K. Moreover, we provide reward and a discounting functions: $R^\times$ and $\gamma^\times$ are a projection of the eventual reward and discounting scheme to the LDBA, as they are only dependent on LDBA states (see Equation 3). A trajectory $\tau = (b_0, b_1, \dots)$ can be sampled from the MRP according to $p(\tau) = \prod_{i=1}^{\infty} K_{b_{i-1}, b_i}$. As any Markov reward process, the newly synthesized one allows computing the value function under eventual discounting $\bar{V}_K(b) = \mathbb{E}_K[\sum_{t=0}^{\infty} \Gamma_t^\times R^\times(b_t)]$ with $b_{t+1} \sim K(b_t)$. The Markov property over the MRP induces the following Bellman equation [37]:

$$\bar{V}_K = R^\times + \gamma^\times \odot (K \bar{V}_K), \tag{4}$$

where a matrix notation is adopted: $\bar{V}_K, R^\times$ and $\gamma^\times$ are represented as $|\mathcal{B}|$-dimensional vectors, and $\odot$ stands for the Hadamard product. As the LDBA (and thus the MRP) is discrete and finite, the Bellman equation defined over the MRP has a closed-form solution [37] [2]:

$$\bar{V}_K = (\mathbb{I} - \gamma^\times \odot K)^{-1} R^\times, \tag{5}$$

where $\mathbb{I}$ represents the identity matrix. As the MRP is closely related to the product MDP $\mathcal{M}^\times$, an analysis of their connection is provided in Appendix B.

Once value estimates $\bar{V}_K$ are computed, they can be treated as a potential function for reward shaping [27], although under eventual discounting [42]:

$$R_{\text{intr}}(b, b') = \gamma^\times(b') \bar{V}_K(b') - \bar{V}_K(b). \tag{6}$$

This reward signal can be added to the product MDP reward $R^\times$ and optimized with an arbitrary RL algorithm. We note that, as an instantiation of potential-based reward shaping, the optimal policy in the product MDP remains invariant to this reward transformation, albeit under one additional assumption.

Proposition 3.1. *(Consistency) Let us consider the product MDP* $\mathcal{M}^\times = (\mathcal{S}^\times, \mathcal{A}^\times, \mathcal{P}^\times, R^\times, \gamma^\times, \mu_0^\times)$ *and its modification* $\widetilde{\mathcal{M}}^\times = (\mathcal{S}^\times, \mathcal{A}^\times, \mathcal{P}^\times, R^\times + R_{intr}, \gamma^\times, \mu_0^\times)$. *Under eventual discounting, any optimal policy in* $\widetilde{\mathcal{M}}^\times$ *for which* $\Gamma_t \overset{t \to \infty}{\to} 0$ *is also optimal in* $\mathcal{M}^\times$.

The proof follows the general scheme for potential-based reward shaping [27] and extends it to the eventual discounting setting (see Appendix A).

[2] In the case of exceedingly complex specifications, and thus large LDBAs, iterative methods could be a viable replacement.

While this procedure allows the synthesis of an intrinsic reward signal for arbitrary logic specifications, the informativeness of this signal relies on two factors. The first factor is the existence of a sink state, which occurs across many (but not all) LDBAs constructed from LTL formulas (e.g., formulas involving global avoidance, as the illustrative task T_0). Its absence can be remedied by augmenting the MRP state space with a virtual sink state, reachable from all other states and associated with an eventual reward and discount factors of 0 and 1, respectively (see Appendix C for a complete discussion and evaluation). The second factor lies in the transition kernel K. While Proposition 3.1 guarantees that no kernel perturbs the optimal policy, it does not quantify how a chosen kernel affects learning efficiency. The following section outlines how a careful choice of its initialization and estimation is crucial to the practical effectiveness of the algorithm. Other alternatives are ablated empirically in Appendix F.

3.2 Optimistic Priors for High Level Value Estimation

Let us consider a naive approach to the choice of the transition kernel K, which simply computes the expected empirical transition probability of the current policy π. As shown at the top of Figure 3 for the illustrative task T_0, a randomly initialized policy can be executed in the product MDP, and K can be estimated through the empirical count of observed transitions in the LDBA. As long as the policy does not spontaneously visit the accepting state, values and rewards estimated through Equations 5 and 6 are uniformly zero and fail to drive exploration. The method would thus be rendered ineffective.

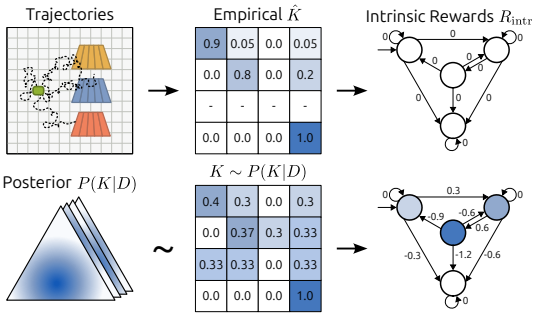

Figure 3: Top: a randomly initialized policy is executed in the product MDP $\mathcal{M}^{\times}$ for `T0`; its empirical transition kernel $\hat{K}$ results in uniformly zero value estimates $\bar{V}_K$. Bottom: the expected value for K over a posterior distribution estimated from a symmetric prior results in informative value estimates and intrinsic rewards.

In order to address this issue, we adopt a Bayesian approach to the problem of estimating the LDBA transition kernel K. Let us consider each row K_i, which models a categorical probability distribution over future LDBA states from each LDBA state $b_i \in \mathcal{B}$. At its core, our method proposes a prior distribution over these categoricals, such that appropriate shaping of the prior controls and directs the exploration in the product MDP. As the conjugate prior to categorical distributions, we adopt a Dirichlet prior $K_i \sim P(K_i) = \mathbf{Dir}(\alpha_{i,0}, \dots, \alpha_{i,|\mathcal{B}|-1})$. The prior is informed of the LDBA structure by setting $\alpha_{i,j} = 0$ if the LDBA does not allow transitions from b_i to b_j. For the m remaining non-zero Dirichlet parameters, we adopt an partially symmetric prior by setting them to a shared value $\frac{\alpha}{m}$, where α is a hyperparameter controlling the strength of the prior: large values of α induce slower convergence of the posterior distribution to the empirical transition kernel.

We remark that the choice of symmetry corresponds to the assumption that, at each step, the agent is capable of transitioning to each adjacent LDBA state with equal probability. In practice, the agent might actually take several steps in the underlying MDP in order to transition to any different LDBA state; moreover, some LDBA transitions can be substantially harder to achieve than others. Finally, for complex problems, naive exploration may not even result in observing the full set of possible transitions in a practical number of time steps. However, assuming that the agent is capable of transitioning under a max-entropy distribution allows reward signals to propagate to all states, thus resulting in informative estimates for the high-level value $\bar{V}_K$ and the intrinsic rewards R_{intr}. For the illustrative example T_0, this is displayed in the bottom part of Figure 3.

Furthermore, while the initialization of K is possibly unrealistic, the Bayesian framework provides a natural way to update it as an N-step trajectory $D = (b)_0^N$ is gathered in the product MDP:

$$P(K_i|D) \propto P(D|K_i)P(K_i) \tag{7}$$

for each $i \in [0, \dots, |\mathcal{B}|]$. This update is tractable due to the choice of a conjugate Dirichlet prior for the categorical likelihood described by the transition kernel. As training progresses, a posterior distribution $P(K|D)$ can be updated with collected evidence. Moreover, instead of computing high-level values for a specific transition kernel K as in Equation 5, we can compute the expected

value $\bar{V}$ under the posterior distribution of transition kernels $P(K|D)$ and thus of MRPs:

$$\bar{V} = \mathop{\mathbb{E}}_{K \sim P(K|D)}[\bar{V}_K] = \mathop{\mathbb{E}}_{K \sim P(K|D)}[(I - \gamma^{\times} \odot K)^{-1}]R^{\times}. \tag{8}$$

We remark that the maximization of $\bar{V}$ corresponds to the average-case MDP problem, while other procedures can be easily adapted to solve the Robust MDP [47] or the Percentile Criterion MDP problems [10]. In practice, the expectation in Equation 8 can be estimated by sampling, with the special case of Thompson Sampling when a single sample is used. Our approach to estimating the transition kernel is ablated empirically in Appendix F; a study of the hyperparameter α controlling prior strength is in Appendix G.

3.3 Practical Algorithm

This section combines the elements outlined above into a practical scheme for distilling an intrinsic reward, which is reported in Algorithm 1. On top of the ability to sample trajectories from the product MDP and access to its LDBA, the algorithm only requires the specification of a prior distribution $P(K)$ over LDBA transition kernels, which is proposed in Section 3.2. The output is a policy π optimizing an eventually discounted proxy to the likelihood of task satisfaction.

Algorithm 1 DRL^2

Input: Product MDP $\mathcal{M}^{\times}$, prior distribution $P(K)$
for each iteration **do**
  Execute policy π in $\mathcal{M}^{\times}$ to collect data D for N steps
  Update posterior $P(K|D)$ with evidence D (Eq. 7)
  Compute high-level values $\bar{V}$ (Eq. 8)
  Sample training batch B (either on- or off-policy)
  Add intrinsic reward to batch B (Eq. 6)
  Train π with B through arbitrary RL algorithm
end for

4 Experiments

This section presents an empirical evaluation of the method by investigating the following questions:

- Can DRL^2 drive deep exploration in reinforcement learning from linear temporal logic specification?
- How does DRL^2 perform across different environments and specifications?
- Can DRL^2 be scaled to high-dimensional continuous settings?

As the method is designed to handle the full complexity of LTL, our evaluation considers several logic specifications, encompassing reach-avoidance (e.g., $Fa \wedge G(\neg b)$) and subtask sequences (e.g., $T_0 : GF(a \wedge XFc) \wedge G\neg b$). In order to focus on exploration, the suite of formulas allows easily scaling the number of LDBA states, or the minimum number of steps required in the underlying MDP to induce a transition in the LDBA. To investigate the final question, we perform an evaluation in both *tabular* and *continuous* domains. While the former avoids confounding effects arising from function approximation, the latter stresses the ability to scale to complex underlying environments. This also evaluates the versatility of DRL^2, as it is in practice coupled with tabular Q-learning [45] and Soft Actor Critic [12], respectively. In the first case, the environment involves navigation in a 2D GridWorld, while in the latter we evaluate dexterous manipulation with a simulated Fetch robotic arm and locomotion of a 12DoF quadruped robot and a 6DoF HalfCheetah. A detailed description of the benchmark environments and specifications is provided in Appendix J; code is available at sites.google.com/view/drl-2.

Baselines We compare DRL^2 to (1) a counterfactual experience replay scheme (LCER [42]), (2) a novel baseline, that relies on inverse square root visitation counts of LDBA states to compute a potential function for reward shaping (see Appendix J), (3) the underlying learning algorithm with no additional exploration bonuses. We note that other promising approaches for exploration in the LTL domain exist, but they rely on meta-learned components or ad-hoc training regimes [31, 44] and are thus not suitable for a fair comparison.

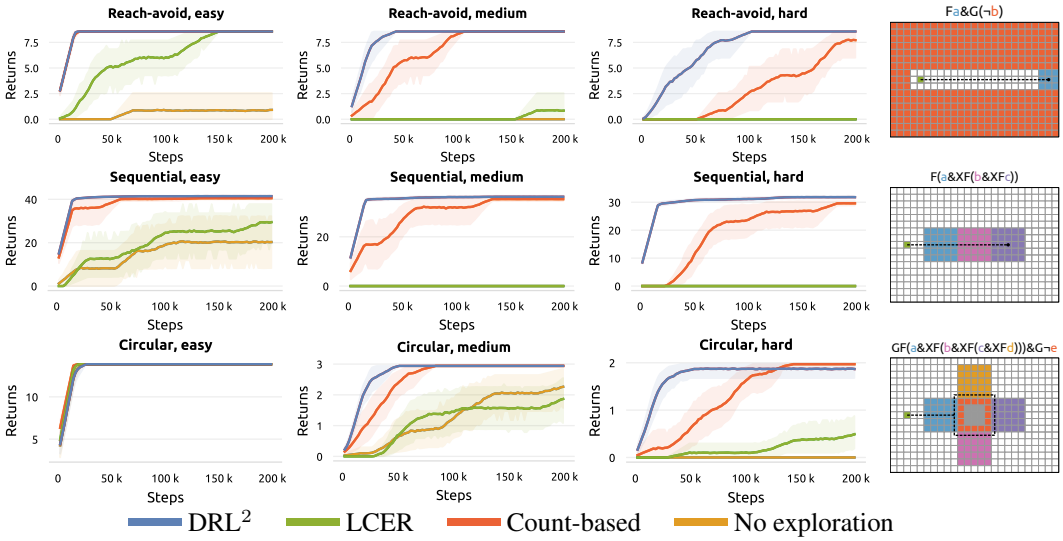

Figure 4: Return for Q-learning under eventual discounting. The three rows display reach-avoidance, sequential and circular tasks, as illustrated on the right with optimal policies. Each atomic proposition evaluates to $\top$ in cells of matching color; difficulty increases from left to right. Further details are available in Appendix J. DRL^2 is able to drive exploration when naive exploration is insufficient.

4.1 Tabular Setting

We first evaluate the exploration performance of DRL^2 by coupling it with tabular Q-learning [45] when operating over discrete state and action spaces in the standard online episodic setting [37]. The evaluation environment is a deterministic 2D gridworld, in which the agent can move in each of the four cardinal directions by one unit at each timestep. The first row of Figure 4 evaluates a reach-avoidance task $G(a \wedge \neg b)$, in which the agent needs to avoid a large area that covers all but a corridor. The goal area is at the end of said corridor; and the difficulty of the task increases with the length of the corridor. In this case the benefit of DRL^2 is evident, as it returns a negative reward whenever the agent leaves the corridor, and the LDBA thus transitions to a sink state. The agent can therefore direct its exploration towards a fraction of the state space, resulting in improved sample efficiency. On the other hand, count-based shaping can only penalize transitions to the sink state once it has been visited enough times. While LCER has the advantage of potentially ignoring failures by hallucinating counterfactual LDBA states during training, it cannot discourage exploration of the forbidden area. We remark that LCER remains a strong method significant when exploration of the underlying MDP is not necessary; a detailed discussion is provided in Appendix E.

The second and third row of Figure 4 evaluate sequential tasks. In both, the agent navigates a room with several zones. In the first case, the agent must reach a sequence of zones in a given order; in the second one, this must be repeated indefinitely while also avoiding the center of the room. While the standard reward under eventual discounting would be zero until the last zone in the sequence is reached, DRL^2 provides an informative reward at each LDBA transition, thus informing the agent to direct exploration towards promising directions. Therefore, as number and size of the zones grows (to the right), DRL^2 results in more efficient exploration by encouraging LDBA transitions towards the accepting state. This encouragement is instead only dependent on visitation counts for the count-based baseline and absent in LCER. We additionally evaluate variations of these tasks in Appendix D.

4.2 Continuous Setting

After verifying the effectiveness of DRL^2 in interpretable settings, this section investigates if the exploration signal can be scaled to high-dimensional, long-horizon environments requiring the application of deep RL algorithms. For simplicity, we evaluate its application to Soft Actor Critic (SAC) [12], which stands as a fundamental building block for many algorithms [11, 13, 18, 24]. On the top of Figure 5 a simulated Fetch robotic arm [9] is evaluated on two tasks, namely (i) moving its gripper to a specific location while avoiding lateral movements and (ii) gradually producing an horizontal alignment of three cubes. In the middle, an HalfCheetah receives specifications encoding, respectively, finite sequences of positions and infinite sequences of angles for its center of mass,

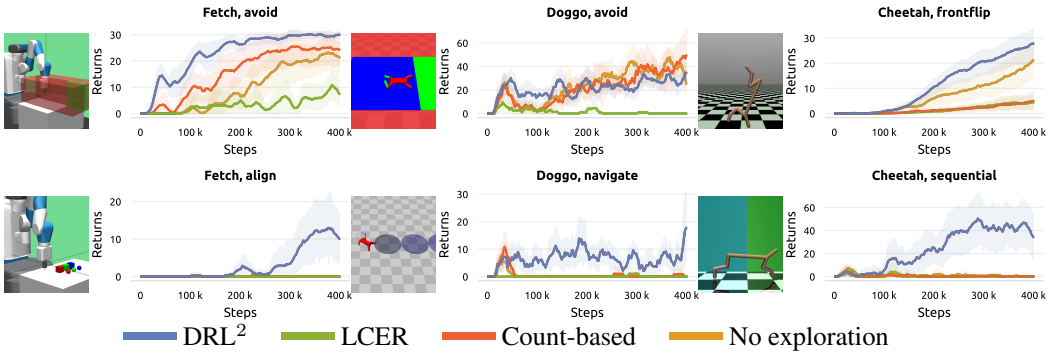

Figure 5: Return for SAC under eventual discounting on Fetch (left), Doggo (center) and HalfCheetah (right), as shown in renderings. DRL^2 confirms its ability to drive exploration in complex tasks when coupled with deep RL.

resulting in precise horizontal locomotion, and in front flipping indefinitely. On the bottom, a 12DoF simulated quadruped robot [33] is tasked with (i) fully traversing a narrow corridor, or with (ii) navigating through two zones in sequence. As in the tabular case, the four tasks can all be described through LTL formulas encoding reach-avoidance and sequential behavior. They are reported among further details on the environments in Appendix J.

As expected, in this setting the evaluation is slightly noisier and partially constrained by the learning algorithm. Nevertheless, when applied to significantly more complex underlying environments, DRL^2 is competitive with the stronger baselines in simpler problems, and largely outperforms them when exploration in the underlying MDP becomes more challenging. Interestingly, we observe that counterfactual experience replay (LCER) is less effective in this setting. We remark that LCER can generate unfeasible states for the product MDP, which can be harmful when the agent has the ability to interpolate between training samples.

5 Related Work

Learning from LTL specification has seen remarkable progress in recent years. While this section provides an essential overview, an extended selection of works is presented in Appendix H.

LTL is among several languages for task specification in RL: for instance, numerous works have investigated Reward Machines [19, 20], which however do not match the expressiveness of LTL. While Reward Machines can be very effective for reward shaping, either through heuristics [5] or value iteration [6], these methods are static and do not naively generalize to our setting.

The combination of LTL and RL has initially focused on reconciling logic and policy optimization through the definition of a product MDP and the design of a reward signal encouraging task satisfaction [3, 5, 6, 15, 16, 22, 26]. This led to the development of principled approaches, proposing schemes that provably and directly optimize a lower bound on the likelihood of formula satisfaction [34, 42]. The sparsity of rewarding schemes has motivated the development of several methods, traditionally resorting to heuristics [26], hierarchical decomposition [4, 21], relabeling [31] or metalearning [40, 44], thus introducing potential suboptimality, off-policiness and additional requirements, respectively.

To the best of our knowledge, our approach is novel in its adaptive and informed estimation of the high-level Markov reward process, resulting in a directed exploration method that retains optimality.

6 Discussion

This work proposes DRL^2, an exploration method for reinforcement learning from LTL specifications. By casting the LDBA encoding the specification as a Markov reward process, we enable a form of high-level value estimation, which can produce value estimates for each LDBA state. These values can be leveraged as a potential function for reward shaping and combined with an informed Bayesian estimate of the LDBA transition kernel to ensure an informative training signal. As a result, DRL^2 accelerates learning when non-trivial exploration of the underlying MDP is necessary for reaching an accepting LDBA state, as is often the case for complex specifications.

Limitations DRL^2 is not designed to directly address certain exploration issues (such as jump transitions). Nonetheless, it can be seamlessly combined with existing experience replay methods that do. As several other exploration schemes, we note that DRL^2 introduces a slight non-stationarity in the reward signal, which needs to be addressed by the underlying learning algorithm.

Outlook This work opens up exciting future directions, including a formal analysis of which reward shaping terms would not only grant consistency, but also maximize sample efficiency. Moreover, as DRL^2 leverages the LDBA structure, its effectiveness is dependent on it. Its applicability can thus further benefit by LDBA construction algorithms that do not return an arbitrary LDBA in its equivalence class, but rather the one which is most suitable for guiding an RL agent.

Having shown how the fundamental problem of sparsity for complex LTL tasks can be addressed by directly leveraging their structure, we believe that the evidence provided in this work further supports reinforcement learning as suitable paradigm for extracting a controller from a logic specification, simply via interaction and learning.

Acknowledgments

We thank Nico Gürtler for precious discussions throughout the project, and the anonymous reviewers for their valuable feedback. Marco Bagatella is supported by the Max Planck ETH Center for Learning Systems. This project has received funding from the Swiss National Science Foundation under NCCR Automation, grant agreement 51NF40 180545. Georg Martius is a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645. We acknowledge the support from the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039B).

References

[1] D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. Littman, D. Precup, and S. Singh. On the expressivity of markov reward. *Advances in Neural Information Processing Systems*, 2021.

[2] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT press, 2008.

[3] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[4] M. Cai, M. Hasanbeig, S. Xiao, A. Abate, and Z. Kan. Modular deep reinforcement learning for continuous motion planning with temporal logic. *IEEE robotics and automation letters*, 2021.

[5] A. Camacho, O. Chen, S. Sanner, and S. McIlraith. Non-markovian rewards expressed in ltl: guiding search via reward shaping. In *Proceedings of the International Symposium on Combinatorial Search*, 2017.

[6] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019.

[7] G. De Giacomo, M. Favorito, L. Iocchi, F. Patrizi, and A. Ronca. Temporal logic monitoring rewards via transducers. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, 2020.

[8] G. De Giacomo, L. Iocchi, M. Favorito, and F. Patrizi. Restraining bolts for reinforcement learning agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[9] R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry. Gymnasium robotics. *github*, 2023. URL `http://github.com/Farama-Foundation/Gymnasium-Robotics`.

[10] E. Delage and S. Mannor. Percentile optimization for markov decision processes with parameter uncertainty. *Operations research*, 2010.

[11] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 2018.

[13] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[14] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. Array programming with numpy. *Nature*, 2020.

[15] M. Hasanbeig, A. Abate, and D. Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[16] M. Hasanbeig, D. Kroening, and A. Abate. Deep reinforcement learning with temporal logics. In *Formal Modeling and Analysis of Timed Systems: 18th International Conference, FORMATS 2020*, 2020.

[17] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 2022.

[18] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 2021.

[19] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, 2018.

[20] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 2022.

[21] K. Jothimurugan, S. Bansal, O. Bastani, and R. Alur. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 2021.

[22] Y. Kantaros. Accelerated reinforcement learning for temporal logic control objectives. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[23] J. Křetínskỳ, T. Meggendorfer, S. Sickert, and C. Ziegler. Rabinizer 4: from ltl to your favourite deterministic automaton. In *International Conference on Computer Aided Verification*, 2018.

[24] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 2020.

[25] B. G. León, M. Shanahan, and F. Belardinelli. Agent, do you see it now? systematic generalisation in deep reinforcement learning. In *ICLR Workshop on Agent Learning in Open-Endedness*, 2022.

[26] X. Li, C.-I. Vasile, and C. Belta. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[27] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. *NIPS 2017 Workshop on Autodiff*, 2017.

[29] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 1977.

[30] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[31] W. Qiu, W. Mao, and H. Zhu. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[32] P. Rauber, A. Ummadisingu, F. Mutz, and J. Schmidhuber. Hindsight policy gradients. In *International Conference on Learning Representations*, 2019.

[33] A. Ray, J. Achiam, and D. Amodei. Benchmarking safe exploration in deep reinforcement learning. `https://github.com/openai/safety-gym`, 2019.

[34] D. Shao and M. Kwiatkowska. Sample efficient model-free reinforcement learning from ltl specifications with optimality guarantees. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 2023.

[35] S. Sickert, J. Esparza, S. Jaax, and J. Křetínskỳ. Limit-deterministic büchi automata for linear temporal logic. In *International Conference on Computer Aided Verification*, 2016.

[36] D. Silver, S. Singh, D. Precup, and R. S. Sutton. Reward is enough. *Artificial Intelligence*, 2021.

[37] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[38] W. Thomas. Chapter 4 - automata on infinite objects. In *Formal Models and Semantics*, Handbook of Theoretical Computer Science, pages 133–191. Elsevier, Amsterdam, 1990.

[39] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. Gymnasium, 2023. URL `https://zenodo.org/record/8127025`.

[40] P. Vaezipoor, A. C. Li, R. A. T. Icarte, and S. A. Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In *International Conference on Machine Learning*, 2021.

[41] C. Voloshin, H. Le, S. Chaudhuri, and Y. Yue. Policy optimization with linear temporal logic constraints. *Advances in Neural Information Processing Systems*, 2022.

[42] C. Voloshin, A. Verma, and Y. Yue. Eventual discounting temporal logic counterfactual experience replay. In *International Conference on Machine Learning*, 2023.

[43] C. Wang, Y. Li, S. L. Smith, and J. Liu. Continuous motion planning with temporal logic specifications using deep neural networks. *arXiv preprint arXiv:2004.02610*, 2020.

[44] J. Wang, H. Hasanbeig, K. Tan, Z. Sun, and Y. Kantaros. Mission-driven exploration for accelerated deep reinforcement learning with temporal logic task specifications. *arXiv preprint arXiv:2311.17059*, 2023.

[45] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 1992.

[46] Z. Xiong, J. Eappen, D. Lawson, A. H. Qureshi, and S. Jagannathan. Co-learning planning and control policies using differentiable formal task constraints. *arXiv preprint arXiv:2303.01346*, 2023.

[47] H. Xu and S. Mannor. Distributionally robust markov decision processes. *Advances in Neural Information Processing Systems*, 2010.

A Proof of Proposition 3.1 and Further Analysis

This section presents a proof for Proposition 3.1 in Section 3, which we also report in its entirety for ease of reference, and further discusses how the intrinsic reward is informed by the task.

Proposition A.1. *(Consistency) Let us consider the product MDP* $\mathcal{M}^\times = (\mathcal{S}^\times, \mathcal{A}^\times, \mathcal{P}^\times, R^\times, \gamma^\times, \mu_0^\times)$ *and its modification* $\widetilde{\mathcal{M}}^\times = (\mathcal{S}^\times, \mathcal{A}^\times, \mathcal{P}^\times, R^\times + R_{intr}, \gamma^\times, \mu_0^\times)$. *Under eventual discounting, any optimal policy in* $\widetilde{\mathcal{M}}^\times$ *for which* $\Gamma_t \overset{t\to\infty}{\to} 0$ *is also optimal in* $\mathcal{M}^\times$.

The proof consists of a simple extension of known results on potential-based reward shaping [27] to account for eventual discounting. Let us consider an arbitrary state-action pair $(s_0, a_0) \in \mathcal{S}^\times \times \mathcal{A}^\times$ and the optimal Q-function in $\widetilde{\mathcal{M}}^\times$ under eventual discounting :

$$\tilde{Q}^\gamma_\star(s_0,a_0) = \mathop{\mathbb{E}}_{\pi^\star,P^\times}\Big[\sum_{t=0}^{\infty}\Gamma^t(R^\times(s_t,a_t,s_{t+1}) + R_{\text{intr}}(s_t,a_t,s_{t+1}))\Big] \tag{9}$$

$$= \mathop{\mathbb{E}}_{\pi^\star,P^\times}\Big[\sum_{t=0}^{\infty}\Gamma^t R^\times(s_t,a_t,s_{t+1}) + \sum_{t=0}^{\infty}\Gamma^t R_{\text{intr}}(s_t,a_t,s_{t+1}))\Big] \tag{10}$$

$$= Q^\gamma_\star(s_0,a_0) + \mathop{\mathbb{E}}_{\pi^\star,P^\times}\Big[\sum_{t=0}^{\infty}\Gamma^t R_{\text{intr}}(s_t,a_t,s_{t+1}))\Big] \tag{11}$$

$$= Q^\gamma_\star(s_0,a_0) + \mathop{\mathbb{E}}_{\pi^\star,P^\times}\Big[\sum_{t=0}^{\infty}\Gamma^t(\gamma^\times(s_{t+1})\bar{V}(b_{t+1}) - \bar{V}(b_t))\Big] \tag{12}$$

$$= Q^\gamma_\star(s_0,a_0) + \mathop{\mathbb{E}}_{\pi^\star,P^\times}\Big[\sum_{t=0}^{\infty}\Gamma^t\gamma^\times(s_{t+1})\bar{V}(b_{t+1}) - \sum_{t=0}^{\infty}\Gamma^t\bar{V}(b_t))\Big] \tag{13}$$

$$= Q^\gamma_\star(s_0,a_0) + \lim_{t\to\infty}\mathop{\mathbb{E}}_{\pi^\star,P^\times}\Big[\Gamma^t\bar{V}(b_t) - \gamma^\times(s_0)\bar{V}(b_0)\Big] \tag{14}$$

$$= Q^\gamma_\star(s_0,a_0) + \lim_{t\to\infty}\mathop{\mathbb{E}}_{\pi^\star,P^\times}\Big[\Gamma^t\bar{V}(b_t)\Big] - \gamma^\times(s_0)\bar{V}(b_0) \tag{15}$$

where $Q^\gamma_\star$ is the optimal Q-function in $\mathcal{M}^\times$ under eventual discounting. If the assumption holds, and $\Gamma_t \overset{t\to\infty}{\to} 0$, then the second term fades, and we have

$$\tilde{Q}^\gamma_\star(s_0,a_0) = Q^\gamma_\star(s_0,a_0) - \gamma^\times(s_0)\bar{V}(b_0) \tag{16}$$

The difference between the two Q-functions $\tilde{Q}^\gamma_\star$ and $Q^\gamma_\star$ is therefore constant at any given state. Thus,

$$\tilde{\pi}^\star(s) = \operatorname*{argmax}_{a\in\mathcal{A}^\times}\tilde{Q}^\gamma_\star(s,a) = \operatorname*{argmax}_{a\in\mathcal{A}^\times}Q^\gamma_\star(s,a) = \pi^\star(s). \quad \square \tag{17}$$

As stated above, this invariance relies on the assumption that the product of discounts converges to zero, which would happen in case an accepting state is visited infinitely often. Intuitively, once a good policy is found, the contribution of the intrinsic reward fades, and there is no incentive for optimization algorithms to update the policy away from its optimum. While this assumption often holds for the optimal policy (e.g., in deterministic environments), there are cases in which this may not be true (e.g., in the stochastic case, if the LDBA features a sink state). Nevertheless, we note that the assumption holds for optimal policies in tasks considered in this paper, and in recent literature [42].

On the other hand, in cases in which the assumption does not hold (e.g., as often happens for a random initialized policy), the intrinsic reward can in principle still bias the policy and drive exploration well. For instance, let us assume that the likelihood of visiting an accepting LDBA state under the current policy π is exactly zero. This implies $R^\times(\cdot) = 0$, $\gamma^\times(\cdot) = 1$ and $\Gamma_i = \prod_{t=0}^{i-1}\gamma^\times(b_t) = 1$. In this case, the value under eventual discounting for an arbitrary state $s_0 \in \mathcal{S}^\times$ is

$$\tilde{V}^{\gamma}_{\pi}(s_0, a_0) = \mathop{\mathbb{E}}_{\pi^\star, P^\times}\Big[\sum_{t=0}^{\infty} \Gamma^t (R^\times(s_t, a_t, s_{t+1}) + R_{\text{intr}}(s_t, a_t, s_{t+1}))\Big] \tag{18}$$

$$= \mathop{\mathbb{E}}_{\pi^\star, P^\times}\Big[\sum_{t=0}^{\infty} R_{\text{intr}}(s_t, a_t, s_{t+1}))\Big] \tag{19}$$

$$= \mathop{\mathbb{E}}_{\pi^\star, P^\times}\Big[\sum_{t=0}^{\infty} \gamma^\times(s_{t+1})\bar{V}(b_{t+1}) - \bar{V}(b_t)\Big] \tag{20}$$

$$= \lim_{t\to\infty} \mathop{\mathbb{E}}_{\pi^\star, P^\times}\Big[\bar{V}(b_t)\Big] - \bar{V}(b_0). \tag{21}$$

Thus, a policy optimizing the value function under eventual discounting with the intrinsic reward provided by DRL^2 also maximizes the likelihood of eventually reaching the LDBA state with maximum high-level value $\bar{V}$. It is easy to show that this corresponds to the accepting state $b^\star$ in the LDBA as the high-level value of any LDBA state $b \in \mathcal{B}$ is $\bar{V}(b) = (1 - P_\pi(\neg b^\star | b))\bar{V}(b^\star) \leq \bar{V}(b^\star)$, where $P_\pi(\neg b^\star | b)$ is the likelihood of never reaching the accepting state under the current policy.

We can thus conclude that a policy trained with DRL^2 preserves its asymptotic optimality, while also benefiting from an informative exploration signal during early stages of training.

B Connection Between MRP and Product MDP

This section discusses the relationship between product MDP (introduced in Section 2) and MRP (described in Section 3) in further detail. Let us consider a trajectory in the product MDP $\tau = ((s_0, b_0), (s_1, b_1), \dots)$. Due to the construction of the MRP, the return of the projection of τ to the MRP state space (in this case, $(b_0, b_1, \dots)$) is the same as the return of τ in the product MDP, evaluated according to $R^\times$ and $\gamma^\times$. A similar argument can be made for a policy π given a fixed starting state (s_0, b_0). If the MRP transition kernel K is a projection of the transition kernel induced by π in the product MDP, then the value $V_K(b_0)$ computed in the MRP matches the value computed in the product MDP $V^\pi((s_0, b_0))$. Nonetheless, we remark that the MRP can be seen as a high-level approximation of the product MDP, and as such it loses information. Thus, it may not be straightforward to leverage the MRP to accurately compute values for arbitrary MDP states, to the best of our knowledge.

C Addition of a Virtual Sink State

As described in Section 3.1, a design choice in DRL^2 involves the construction of a virtual sink state for LDBAs that do not feature one, where a sink state can be defined as an LDBA state $b_s \in \mathcal{B}$ such that $P^{\mathcal{B}}(b_s, \cdot) = b_s$.

Let us consider an LDBA featuring a path from each state to an accepting state. Moreover, let us consider a transition kernel K with full support over reachable LDBA states (as is the case for likely samples under the prior proposed in Section 3.2). In this case, the values computed under eventual discounting by solving Equation 8 would be uniform: $\bar{V}_K = \frac{1}{1-\gamma}$. As a consequence, intrinsic rewards computed through Equation 6 would be constantly zero until an accepting state is visited and thus uninformative. This is a natural consequence of the fact that, if irreversible failure is not possible, any policy inducing a transition kernel with a good enough support will satisfy the specification, considering an infinite horizon.

In order to provide an informative learning signal, DRL^2 augments LDBAs that do not feature a sink state with a virtual one and assumes that it can be reached from any other LDBA state. We note that this is equivalent with associating each LDBA state with a discount factor equal to the likelihood of transitioning to the virtual sink. While this can introduce myopic behavior [42], we remark that DRL^2 also learns and updates its estimate of the LDBA transition kernel. As any virtual sink state cannot in practice be reached, the likelihood of transitioning to it from each visited LDBA state converges to zero as the number of environment steps increases. Thus, as training progresses, any virtual sink is effectively removed, together with any degree of myopia introduced.

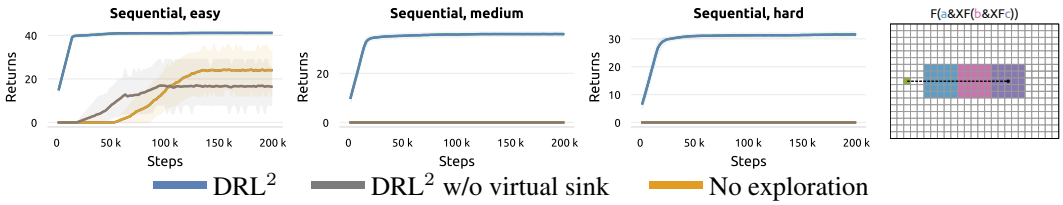

Figure 6: Ablation of the addition of a virtual sink for DRL^2. When a sink state is not featured nor added (e.g. in sequential tasks), the performance of the No-exploration baseline is recovered.

Among the tasks considered in Figure 4, MRPs for sequential tasks do not naturally feature a sink state. To support the discussion, we additionally provide empirical evidence on the performance of DRL^2 in these tasks when a virtual sink is not added. We note that performance on other tasks would not be affected, as they naturally feature a sink state.

In Figure 6 we observe that DRL^2 without a virtual sink state recovers the performance of the No-exploration baseline, as expected. Experimental settings are the same as those outlined in Section 4.

D Additional Tabular Results

Optimal policies for the reach-avoidance tasks from Figure 4 (first row) would also solve a shortest-path reaching problem in the underlying MDP. In other words, a linear path to the goal does not violate the constraints. This Section evaluates a variation of the original task, by reshaping the safe area of the reach-avoid task into a U-shape. In this case, the shortest path to the goal violates the avoidance constraints. Fortunately, changing the shape of the avoidance zone only results in a local change to the labeling function $\mathcal{F}$. DRL^2, as well as other baselines, can handle arbitrary relabeling functions; thus these methods can adapt to arbitrary environment layouts.

Results are reported in 7. for all task variants (easy, medium and hard). In general, as the path length has now increased, performance globally decreases accordingly. Nevertheless, the overall relative performance of each methods is consistent with the original version of the task. We note that LCER [42] now outperforms the count-based baseline, as it can effectively ignore previous visits to the zone to avoid. In the extreme case in which the length of the U-maze is very large, but the starting position is very close to the goal if the avoidance constraint is ignored, we expect LCER to perform even better.

E Comparison to LTL-guided Counterfactual Experience Replay (LCER)

The environments and tasks presented in Section 4 require significant exploration in the underlying MDP and stress the ability to leverage the semantics of LDBA states to accelerate this process.

LTL-guided Counterfactual Experience Replay [41] takes an orthogonal approach to the problem of exploration when learning from LTL formulas and represents a very strong baseline in specific environments. In particular, LCER does not leverage the semantics of the LDBA, however it generates synthetic experience by fully controlling its dynamics. Transitions $((s,b),a,(s',b'))$ with $\{(s,b),(s',b')\} \subseteq \mathcal{S}^{\times}$ and $a \in \mathcal{A}^{\times}$ are relabeled by uniformly resampling the initial LDBA state and simulating the LDBA transition caused by action a, resulting in a new transition $((s,\hat{b}),a,(s',\hat{b}'))$.

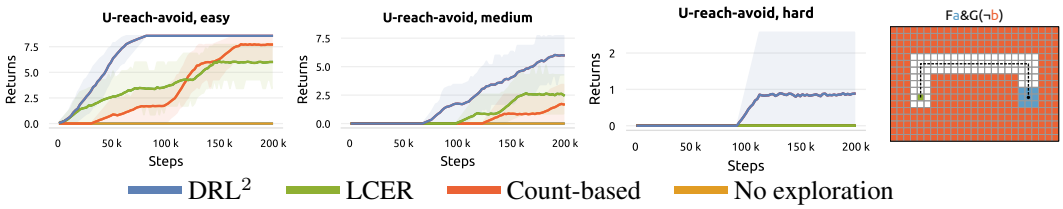

Figure 7: Alternative evaluation of reach-avoid task involving a conflict between reaching objectives and avoidance constraints. Results are consisted with those in Figure 4.

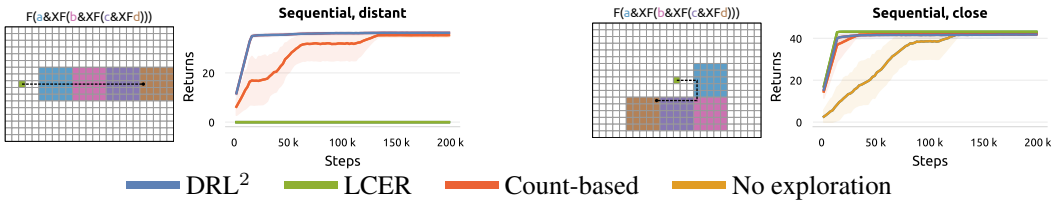

Figure 8: Comparison of DRL^2, LCER and additional baselines in the Sequential task from Figure 4 (left) and an additional variation (right). LCER performs very well when significant exploration of the underlying MDP is not required, as is the case for the example on the right. To the left of each set of training curve, the task is represented, and an optimal policy is visualized.

Generated transitions respect the transition kernel of the product MDP and are thus valid training samples. However, the method does not guarantee that relabeled product states $(s, \hat{b})$ are actually feasible, as the set of reachable product states might only consist of a manifold of the full Cartesian product of MDP and LDBA state spaces. For instance, let us consider the illustrative task T_0 as displayed in Figure 2: at any step, if the agent is in the red field, the LDBA would necessarily be in its sink state and in no other state. Any relabeling of this particular state would thus not be relevant. Furthermore, as a form of state relabeling, LCER cannot generate on-policy data, as the on-policy action sampled in $(s, \hat{b})$ might be different than the original action sampled in (s, b) [32].

Nevertheless, LCER remains a very strong exploration method in environments in which each MDP state can be associated with several LDBA states; furthermore, in the presence of sink LDBA states, LCER can hallucinate their avoidance. In this merit, we report an additional experiment in Figure 8, evaluating DRL^2, LCER and baselines in a modified version of the Sequential environment from Figure 4. In its original version (on the left), the accepting LDBA state can only be visited once the distant final zone (in bright purple) is reached. In this case, LCER is not effective. However, if the zones are rearranged in a way such that they can be visited without excessive exploration in the underlying MDP (e.g., by placing them closer to the starting MDP state, as done on the right), the performance of LCER largely improves, fully bridging the gap with DRL^2. We note that the modified version of the task requires fewer steps to reach an accepting state from the starting one; the baseline without exploration incentives also solves the task, albeit less efficiently. For evaluations on additional variations of tabular tasks, we refer the reader to Appendix D and I.

LCER and DRL^2 thus address different issues in exploration from logic formulas. Fortunately, the two methods are orthogonal and can be freely combined.

F Kernel Estimation Ablation

Proposition 3.1 states that, under specific assumptions, optimal policies are invariant to the proposed reward shaping. In fact, Proposition 3.1 holds for any choice of transition kernel K. In practice, the choice of kernel still impacts the quality of exploration and sample efficiency. We ablate this choice by comparing the effectiveness of kernels sampled by our methods with several others, to assert what constitutes a "good" kernel for exploration.

We remark that DRL^2 averages its values over transition kernels sampled from a posterior distribution to a symmetric Dirichlet prior. We compare this choice to three additional ones:

- kernels sampled from a partially informed prior, which differs from the one used in DRL^2 due to its lack of knowledge over the connectivity of the MRP/LDBA, except for its sink state. It essentially assumes that a transition between any LDBA state pair is always possible.
- a transition kernel obtained using value iteration (VI), and that therefore represents the optimal policy under assumption of full controllability of an high level MDP. It represents a naive implementation of the reward shaping introduced in Icarte et al. [20]. This kernel has the lowest entropy possible, as value iteration outputs a deterministic policy.
- an empirical kernel, which simply counts LDBA transitions performed by the policy in the product MDP. This kernel does not leverage the LDBA structure, and is visualized in Figure 3.

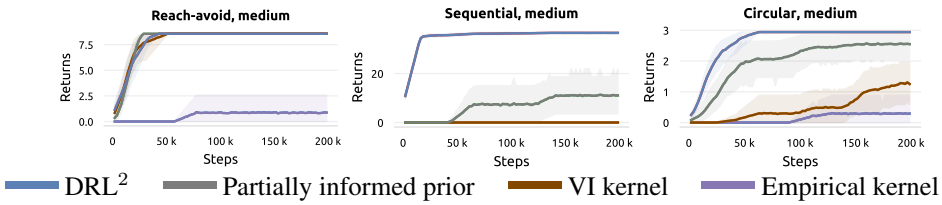

Figure 9: Ablation of kernel initialization and estimation methods. Results suggest that an informed symmetric prior, as used by DRL^2, performs reliably.

These ablations are evaluated in the medium variants of the three tabular tasks from Figure 4 to highlight performance differences; all experimental parameters are unchanged.

Results are presented in Figure 9. We observe that a partially informed prior can recover part of DRL^2's performance, but suffers as the size of the LDBA increases (Sequential, medium). The transition kernel obtained through VI performs well in some tasks, and less in other: since the optimal policy under eventual discounting assigns similar values to non-sink LDBA states, reward shaping becomes less informative. Finally, choosing the empirical kernel generally leads to low performance, as discussed in Figure 3.

G Prior Strength Ablation

This Section performs an empirical evaluation of the effect of the hyperparameter α, which controls the strength of the prior over MRP transition kernels in DRL^2. We ablate the default choice ($\alpha = 1000$) in the hard variant of tasks from the tabular evaluation in Figure 4. Overall, in Figure 10 we observe that DRL^2 is fairly robust to this hyperparameter. Performance is generally lower for weak priors ($\alpha < 100$), which can be significantly affected by poor trajectories collected by the initial policy. As the prior is informed by the LDBA structure, we find that it generally provides good guidance even for stronger priors. We note that, in case the prior is misinformed, a weaker prior would indeed be favorable.

H Extended Related Works

Logic for Task Specification The intersection of reinforcement learning and logic languages has flourished around the topic of Reward Machines [6, 20, 40]. While Reward Machines can handle regular expressions, LTL is designed to address a superset, namely ω-expressions. As a consequence, LTL gains the ability to reason about infinite sequences and tasks including liveness and stability.

For this reason, several works have investigated LTL for task specification. Early attempts proposed heuristic reward shaping schemes [26], which gradually evolved towards dynamic or eventual rewarding and discounting schemes [4, 7, 8, 15, 16, 42]. As for the optimization of such schemes, while linear programming is sufficient for known dynamics and finite action spaces, Q-learning approaches have been adopted when a model of the environment is not available [3, 4, 15]. A fundamental restriction of this approaches was lifted by adopting actor-critic [16] or policy gradient [42] methods suited for continuous action spaces. Nevertheless, in this case, empirical demonstration of LTL-guided RL have remained mostly constrained to low-dimensional setting.

Logic-driven Exploration in Reinforcement Learning A significant effort has been targeted at addressing the sparsity arising from proposed rewarding schemes [15, 42].

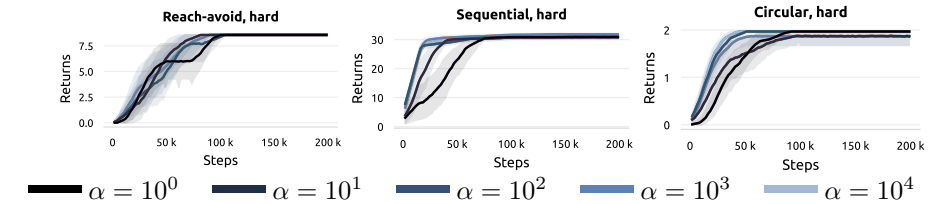

Figure 10: Ablation of prior strength hyperparameter α.

Perhaps closest to the method we propose, several works investigate reward shaping for reinforcement learning from logic specification. In the context of reward machines, Camacho et al. [5] consider potential functions $\Phi(s, b)$ of both the MDP and automaton states; among those, they investigate counting the minimum number of steps to reach an accepting automaton state, which is equivalent to computing values under a specific automaton transition kernel.

Another work [20] explores an extension, which is also fundamentally related to our method: the authors investigate value iteration on an MDP synthetised from the automaton to compute a potential function for reward shaping, thus leveraging the semantics of the task. While motivated by a similar intuition to ours, a naive application of this method would not be viable in an eventually discounted setting, which involves ω-regular problems, as we report in Appendix F. By adopting an LDBA instead of a DFA, and more importantly by integrating an eventual discounting scheme, our method is able to avoid myopic behavior while handling the full complexity of LTL. Futhermore, DRL^2 does not require assumptions on controllability by defining a Markov reward process instead. Values estimated in the MRP are thus softer than the optimal ones in a corresponding MDP, and we find them to be more informative for exploration. We validate this discussion empirically, by evaluating optimal transition kernels under eventual discounting in Appendix F. By updating its distribution over MRP transition kernels, it is moreover capable of providing an adaptive and informative reward signal, rather than a static one derived from a hard assumption.

Other reward shaping techniques in the literature involve the use of an accepting frontier function [15, 16], of bonuses for first visitations [4], or on the knowledge of additional information such as annotated maps [46]. In particular, Hasanbeig et al. [16] tracks and expands the frontier of visited LDBA states, while preventing visits to sinks, but is not task-informed. Our method, in contrast, naturally recovers these two incentives, but biases the first one towards accepting states. Crucially, it also guarantees that the optimal policy is preserved.

Directly within the LTL literature, another line of work leverages the discrete structure of automata and learns hierarchical [16, 20, 21], goal-conditioned [31] or modular [4] policies. While considerably improving sample efficiently, these approaches are known to potentially introduce suboptimality [20].

A further possibility is that of leveraging control over the automaton to produce synthetic experience by relabeling the LDBA states of collected transitions. Such schemes have been proposed in the context of Reward Machines [20] and LTL [42]; as they can produce off-policy samples, they cannot be naively applied to on-policy methods, as discussed in Appendix E. Other relabeling techniques only target the initial LDBA state [43] or focus on optimality guarantees [34].

Finally, we note that previous work have also led to strong exploration on unseen tasks, as the result of a meta-learning phase [25, 40].

I Extension to Jump Transitions

This section extends several passages across Sections 2 and 3 to address the presence of jump transitions in the LDBA, and confirms this extension empirically. Jump transitions occur in a subset of automata, in particular for stability specification.

The definition of LDBA in Section 2 can therefore be extended as follows:

Definition I.1. (Limit Deterministic Büchi Automaton - LDBA) An LDBA is a tuple $\mathcal{L} = (\mathcal{B}, \Sigma \cup \mathcal{A}^{\mathcal{B}}, P^{\mathcal{B}}, \mathcal{B}^{\star}, b_0)$, where $\mathcal{B}$ is a finite set of states, $\Sigma = 2^{AP}$ is an alphabet over atomic propositions, $\mathcal{A}^{\mathcal{B}} : \mathcal{B} \to \{0, \dots, N-1\}$ is the subset of indexed jump transitions available at each state, $P^{\mathcal{B}} : \mathcal{B} \times (\Sigma \cup \mathcal{A}^{\mathcal{B}}) \to \mathcal{B}$ is a transition function, $\mathcal{B}^{\star} \subseteq \mathcal{B}$ is a set of accepting states, and $b_0 \in \mathcal{B}$ is the initial state. There exists a mutually exclusive partitioning of $\mathcal{B} = \mathcal{B}_D \cup \mathcal{B}_N$ such that $\mathcal{B}^{\star} \subseteq \mathcal{B}_D$ and for $(b, a) \in (\mathcal{B}_D \times \Sigma)$ then $P^{\mathcal{B}}(b, a) \in \mathcal{B}_D$. Moreover, $\mathcal{A}^{\mathcal{B}}(b) = \varnothing$ for all $b \in \mathcal{B}_D$ and $P^{\mathcal{B}}(b, a) \in \mathcal{B}_D$ for all (b, a) in $\mathcal{B}_{\mathcal{N}} \times \mathcal{A}^{\mathcal{B}}(b)$.

Similarly, a *path* over an LDBA needs can be induced by an infinite sequence of LDBA *actions* $(a_i)_0^{\infty} \in (\Sigma \cup \mathcal{A}^{\mathcal{B}})^{\infty}$, thus also accounting for jumps.

Finally, the definition of product MDP can also be altered accordingly, as $\mathcal{M}^{\times} = (\mathcal{S}^{\times}, \mathcal{A}^{\times}, \mathcal{P}^{\times}, R^{\times}, \gamma^{\times}, \mu_0^{\times})$, where $\mathcal{S}^{\times} = \mathcal{S} \times \mathcal{B}, \mathcal{A}^{\times} = \mathcal{A} \cup \mathcal{A}^{\mathcal{B}}$, and $\mu_0^{\times}(s, b) = \mu_0(s) \cdot \mathbf{1}_{b=b_0}$. Its

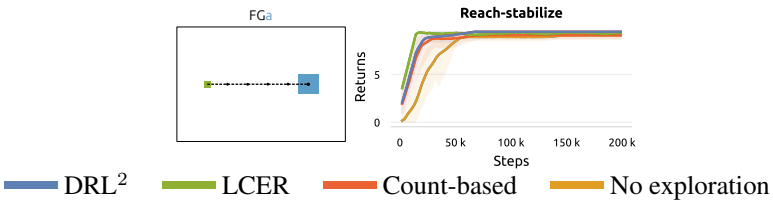

Figure 11: Evaluation of DRL^2 and baselines on a the task FGa in presence of jump transitions.

transition kernel is then defined as

$$P^{\times}((s',b')|a,(s,b)) = \tag{22}$$
$$\begin{cases} P(s'|a,s), & a \in \mathcal{A}, b' = P^{\mathcal{B}}(b, \mathcal{F}(s')) \\ 1 & a \in \mathcal{A}^{\mathcal{B}}(b), b' = P^{\mathcal{B}}(b,a), s = s' \\ 0 & \text{otherwise} \end{cases}$$

The rest of the content through the main body of the paper is already compatible with the existence of jump transitions and does not need extensions.

As discussed above, our methods, as well as all baselines, are in principle compatible with the presence of jump transitions. Thus, we now evaluate the task FGa in reach-avoid tabular settings. This setup is similar to that of the corresponding experiment in recent work [42]. Overall, as the structure of the LDBA is minimal, we observe that all considered methods perform reasonably well in Figure 11. In particular, DRL^2 and the count-based baseline slightly improve over the No-exploration baseline. LCER [42] works particularly well, as the task does not involve several substeps, but presents irrecoverable failures when jumping at the wrong time. We refer to Appendix E for a further comparison between DRL^2 and LCER.

J Implementation Details

This section presents a thorough description of methods, tasks and hyperparameters. For the sake of reproducibility, we provide our full codebase [3].

J.1 Environments and Tasks

Throughout the paper, experiments involve simulated environments.

J.1.1 Tabular Environments

The environments considered in tabular settings (Figure 4) are variants of the common gridworld, that is a 2D grid, with one cell being occupied by the agent. The action space is discrete and includes 5 actions, four of which allow movement in each of the cardinal directions, and a single no-op action. The observation space contains x and y coordinates of the agent and is therefore 2-dimensional. While dynamics are shared, the labeling function from MDP states to atomic propositions and task-dependent details differ across tasks and are reported below. We note that each task is also represented in the rightmost column of Figure 4, although not to scale for illustrative purposes.

Reach-avoidance Reach-avoidance is evaluated in a gridworld without obstacles (Figure 4, top). The agent is initialized at one end of a thin corridor (of unit width). The other end of the corridor is not bounded; however, the AP a evaluates to `true` in each cell of the corridor beyond a fixed distance from the starting cell. The AP f evaluates to `true` anywhere outside the corridor. The formula evaluated is $Fa \wedge G\neg b$. The difficulty of the task can be scaled by increasing the distance to the zone to reach (7, 9 and 11 for easy, medium and hard task variants, respectively). Episodes have a duration equal to the minimum number of step to reach the final zone, plus 10 steps.

[3] sites.google.com/view/drl-2

Sequential The second task in Figure 4 also takes place in a gridworld without obstacles. Contiguous 7×7 squares are aligned horizontally. The agent starts at the center of the leftmost square; in each square to the right, a different AP evaluates to `true`, in alphabetical order. In the first square to the right of the starting one, a evaluates to `true`, in the second b evaluates to `true` and so on. Difficulty can be scaled by specifying longer formulas: $F(a \wedge XF(b \wedge XFc))$, $F(a \wedge XF(b \wedge XF(c \wedge XFd)))$ and $F(a \wedge XF(b \wedge XF(c \wedge XF(d \wedge XFe))))$ are used, respectively, for easy, medium and hard tasks. Episodes have a fixed length of 70 steps.

Circular Circular tasks (Figure 4, bottom) present 5 contiguous 7×7 squares, aligned as a cross. The central zone represents an obstacle: it is labeled to the AP e, and the agent cannot reach its central 6×6 cells. The other 4 zones are labeled as a, b, c, d in counterclockwise order, and the agent is initialized in proximity of the first one. The difficulty of the task can be controlled by scaling the number of zones involved in a loop: $GF(a \wedge XFb) \wedge \neg e$, $GF(a \wedge XF(b \wedge XFc) \wedge \neg e$ and $GF(a \wedge XF(b \wedge XF(c \wedge XFd)) \wedge \neg e$ for easy, medium and hard tasks, respectively. An episode lasts for 70 steps.

J.1.2 Continuous Environments

Section 4 also provides an evaluation on continuous, high-dimensional environments. Since evaluation of LTL-guided RL has traditionally been mostly restricted to simple tabular [20] or low-dimensional [16, 42] settings, we repurpose high-dimensional environments from safe and goal-conditioned reinforcement learning research.

Fetch Experiments reported in the first row of Figure 5 were developed on top of the common Fetch robotic benchmark [9], and are also available as part of our codebase. We evaluate two tasks, in which the agent controls the end effector position of a 7DoF robotic arm with 4-dimensional actions over episodes of 50 steps. `FetchAvoid` is reminiscent of the reach-avoidance task in tabular settings, and involves fully stretching out the arm while avoiding lateral movements (i.e., not entering red zones as shown in Figure 5). In this case, observations are 10-dimensional. In `FetchAlign` the agent has to interact with three cubes on one side of the table, and has to align them horizontally at the center of the table. In this case, observations include information on the cubes, and are 45-dimensional. The two tasks are specified, respectively, with the formulas $Fa \wedge G\neg e$ (reach the end of the table, and avoid lateral movements), and $F(a \wedge XF(b \wedge XFc))$ (position the first, second and then third block).

Doggo Doggo is a 12DoF quadruped adapted from the most challenging tasks in SafetyGym [33]. It is tasked with navigating a flat plane; observations are 66-dimensional, actions are 12-dimensional and the length of each episode is 500 steps. Similarly to other reach-avoidance tasks, `DoggoAvoid` involves navigation to a distant goal (shown in green in Figure 5), in a straight line, while completely avoiding detours. On the other hand, `DoggoLinear` involves navigating through a sequence of 2 circular zones. These two tasks are encoded through the following two specifications: $Fa \wedge G\neg e$ and $F(a \wedge XFb)$. Episode length is set to 500.

HalfCheetah HalfCheetah [39] involves controlling a 6DoF robot in a vertical 2D plane and is a standard environments in the deep reinforcement learning literature. We define two tasks in this underlying environments. In `CheetahSequential` the agent is tasked with eventually reaching, in succession, 4 consecutive zones to its right, as encoded by the formula $F(a \wedge XF(b \wedge XF(c \wedge XFd)))$. Once the last zone is reached, the task is solved and the agent can stop. In `CheetahFrontflip` the agent is informed by the formula $GF(a \wedge XF(b \wedge XF(c \wedge XFd)))$, where each variable evaluates to `true` for a given range of angles of the main body of the robot, respectively when the Cheetah is in its default position, standing on its front legs, upside down, and standing on its back legs. As a result, the formula is satisfied by an infinite sequence of frontflips. Notably, this task can not be satisfied by finite trajectories.

J.2 Methods and Algorithms

The core of our method computes an intrinsic reward and is therefore applicable on top of arbitrary reinforcement learning algorithms. Therefore, we will first discuss the implementation of the method itself, as well as relevant baselines, and then details involving RL algorithms.

The proposed algorithm is detailed in Algorithm 1. Its computational cost is dominated by the closed form solution of the value in the Markov reward model (see Equation 8), which is cubic in the number of MRM states. However, since LDBAs for most common tasks in these settings are relatively small (< 20 states), the computational load is negligible, even when computing the expected value over the posterior via sampling.

The main hyperparameter introduced by DRL^2 is α, which controls the strength of its prior as described in Section 3.2. It is set to 10^3 in tabular settings, and to 10^5 in continuous settings, where increased stability was found to be beneficial. The remaining hyperparameters for reward shaping are shared with the count-based baseline and are, respectively, the frequency of updates to the potential function (set to 2000 environment steps), and a scaling coefficient to the intrinsic reward, which was tuned individually for each method in a grid of $[0.1, 1.0, 10.0]$. We found a coefficient of 0.1 to be optimal across all tasks for both methods in tabular settings, while continuous settings benefit from a stronger signal and a coefficient of 10.0. The two remaining baselines (relying on LCER and no exploration techniques) introduce no additional hyperparameters on top of the learning algorithm.

Table 1: Hyperparameters for Q-learning and Soft Actor Critic.

Hyperparameter	**Value**
γ	0.99
Buffer size	$4 \cdot 10^5$
Batch size	64
Initial exploration	$2 \cdot 10^3$ steps
τ (Polyak)	$5 \cdot 10^{-3}$ for SAC
Learning rate	$3 \cdot 10^{-4}$ for SAC, 1 for Q-learning

The intrinsic reward is added to the extrinsic one and optimized under eventual discounting through one of two algorithms, depending on the setting. In practice, we rely on tabular Q-learning [45] with an ϵ-greedy policy (ϵ=0.1) and Soft Actor Critic with automatic entropy tuning [13]. In the tabular case, strict adoption of eventual discounting [42] results in a largely uniform policy, as in most states no action would reduce the likelihood of task satisfaction. As a consequence, a prohibitive number of steps would be required during evaluation for computing cumulative returns. For this reason, our practical implementation of tabular Q-learning *does not* apply eventual discounting, although this would be possible given a much larger computational budget. Our SAC implementation is partially based on that provided by Huang et al. [17]. Hyperparameters for each algorithm were tuned to perform best when no exploration bonus is being used, and are reported in Tables 1. They are kept fixed across tasks.

J.3 Metrics

Unless specified, the metric used to measure the methods' performance is cumulative return under eventual discounting, that is $R = \sum_{i=0}^{N} \gamma^i$, where $\gamma = 0.99$ and N is the number of visits to an accepting LDBA state within an episode. In the limit of $N \to \infty$, this metric is a proxy for the likelihood of task satisfaction [42]. All curves report mean performances estimated across 10 seeds, and shaded areas represent 95% simple bootstrap confidence intervals. They undergo average smoothing with a kernel size of 10.

J.4 Tools

Our codebase[4] mostly relies on `numpy` [14] for numeric computation and `torch` [28] for its autograd functionality. Furthermore, we partially automate the synthesis of LDBAs from LTL formulas through `rabinizer` [23].

J.5 Computational Costs

All methods have comparable runtimes within their setting (tabular or continuous). Each experimental run required 8 cores of a modern CPU (Intel i7 12th Gen CPU or equivalent). Each run in tabular and continuous settings required on average 30 minutes and 150 minutes, respectively.

[4] Available at sites.google.com/view/drl-2.