Certified Reinforcement Learning with Logic Guidance

Hosein Hasanbeig[a,*], Daniel Kroening[b,*], Alessandro Abate[c,**]

[a] *Microsoft Research*
[b] *Magdalen College, University of Oxford, and Amazon, Inc.*
[c] *Department of Computer Science, University of Oxford, Parks Rd, OX1 3QD, UK*

Abstract

Reinforcement Learning (RL) is a widely employed machine learning architecture that has been applied to a variety of control problems. However, applications in safety-critical domains require a systematic and formal approach to specifying requirements as tasks or goals. We propose a model-free RL algorithm that enables the use of Linear Temporal Logic (LTL) to formulate a goal for unknown continuous-state/action Markov Decision Processes (MDPs). The given LTL property is translated into a Limit-Deterministic Generalised Büchi Automaton (LDGBA), which is then used to shape a synchronous reward function on-the-fly. Under certain assumptions, the algorithm is guaranteed to synthesise a control policy whose traces satisfy the LTL specification with maximal probability.

Keywords: Reinforcement Learning, Control Synthesis, Policy Synthesis, Formal Methods, Temporal Logics, Automata, Markov Decision Processes.

1. Introduction

Reinforcement Learning (RL) is an area of machine learning, where an agent is trained to maximise a reward that is calculated by a user-provided function. Key to success in RL is the ability to predict the effect of picking a particular candidate action on the ultimate reward, and neural networks, owing to their ability to generalise, have enabled the application of RL in a broad range of application domains.

A significant barrier to successful application of RL is the setup of the reward function when requirements (the user's goals or the task that is to be done) are complex [46]: reward engineering often requires tedious parameter tuning to map complex goals to an appropriate reward structure [16]. As a consequence, the trained agent can be brittle and the policy it implements can be difficult to interpret. This is particularly problematic in safety-critical applications, say

*The work reported in this paper was done at Department of Computer Science, University of Oxford, UK.
**Corresponding author
Email address: alessandro.abate@cs.ox.ac.uk (Alessandro Abate)

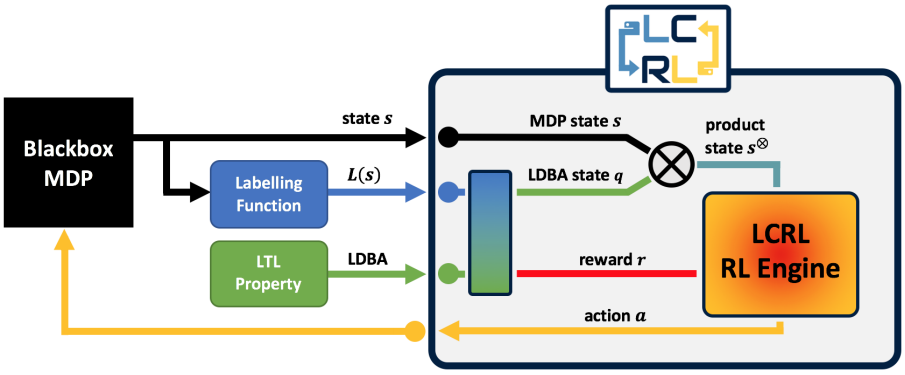


Figure 1: Learning under LTL with Logically-Constrained RL (LCRL). The reward signal is produced automatically by synchronising the LTL property and the unknown MDP.

when the agent operates in the proximity of humans. This gives rise to the need for provably-correct reward shaping.

We suggest the use of Linear Temporal Logic (LTL) [111] as a formal high-level language to specify complex tasks when applying RL [32] to sequential decision-making problems. LTL is a formal language that can express engineering requirements and specifications, and there exists a substantial body of research on how to derive LTL properties from natural language [104, 152, 52, 151]. LTL can express time-dependent properties, such as safety and liveness, and further allows to specify tasks with complex dynamics (e.g., tasks that are repeating, sequential, or conditional).

We present an algorithm that, given an LTL formula that describes the goal the user wants to achieve, automatically shapes a reward function for RL in a provably-correct manner, and synthesises controllers for which satisfaction of the given property is guaranteed.

The standard means for formalising sequential decision-making problems are Markov Decision Processes (MDPs), a family of stochastic processes [115]. In an MDP, a decision maker (or an agent) can transition between states by choosing relevant actions [142] while receiving a scalar reward. The outcomes of taking actions are, in general, probabilistic and not fully under the control of the agent [15]. A decision-making problem given as an MDP is said to be solved when in any given state the agent is able to choose the most favourable action so that the accrued reward is expected to be maximal in the long run [142].

When both state and action spaces are finite, the stochastic behaviour of an MDP can be described by a transition probability matrix. In this case, MDPs can be solved via Dynamic Programming (DP). DP iteratively applies a Bellman operation on a value function expressing the expected reward of interest for a state of the MDP [11, 15]. We can understand convergence properties of RL by relating them to the optimal value function produced by DP [147, 72]. When the state and action spaces are not finite, approximate DP is often employed. This

approximation can be achieved by generating an abstract model of the MDP itself [13, 113, 38, 130], or by inferring a (non-linear) regression model over the value function [123, 102, 131].

In practice, however, it is often infeasible to obtain sufficient knowledge of the stochastic dynamics of the problem, which means that we cannot formulate the MDP. In contrast to DP, RL solely depends on a set of experience samples gathered by exploration, which simplifies the Bellman's backup, and at the same time avoids exhaustive sweeps over the full state space. In this paper, we focus on approaches that are *model-free*, i.e., we do not require that any model (in the form of an MDP or otherwise) is given. Model-free RL methods are very easy to apply and resource-efficient, as the agent learns an optimal policy without an intermediate model. Model-free RL has been shown to converge to the same action selection policy as DP under mild assumptions [15, 147].

Deep RL is arguably one of the most significant breakthroughs in RL, whereby human-level play has been achieved on a number of Atari games [101] by incorporating deep neural networks into RL. This resulted in successfully tackling StarCraft [144] and the game of Go [128]. Deep-RL-based algorithms are often general enough so that the rules of different games do not have to be explicitly encoded for the agents to learn successful control policies. The success of deep RL has resulted in extensive use of RL, beyond small-scale environments [128, 144, 67, 51]. In particular, employment of RL in safety-critical problems has recently been investigated [33, 47, 45, 106, 58, 3, 73, 59, 30], including autonomous driving [124, 122] and avionics [96, 2]. This however inevitably entails the need for correct-by-design policy synthesis, in order to guarantee for instance, among other quantitative requirements, the safety of policies synthesised via RL.

Contributions: We present an algorithm for automatically engineering a reward function for RL from a given LTL property, and for synthesising a corresponding optimal policy maximising the probability of satisfying the given LTL property. Our method is model-free, and allows us to synthesise control policies under LTL for a continuous-state/action MDP (which subsumes the simpler, finite state/action special case). We discuss the assumptions under which our RL setup is guaranteed to synthesise control policies whose traces satisfy the LTL specification with maximal probability.

The LTL property offers means to introduce a-priori knowledge of the structure of the problem into the learning procedure, while avoiding overfitting demonstrations done by a human teacher. An LTL property is a formal, ungrounded, and symbolic representation of the task and of its steps. This enables the use of complex properties and, as we show later, LTL is suitable for sub-task decomposition and hierarchical learning.

In existing methods, the temporal dependency of rewards is often tackled with *options* [132], or, in general, dependencies are structured hierarchically. Current approaches to hierarchical RL very much depend on particular state representations and whether they are structured enough for a suitable reward signal to be effectively engineered manually. Hierarchical RL therefore often requires detailed supervision in the form of explicitly specified high-level actions

or intermediate supervisory signals [114, 78, 34, 81, 143, 6, 8]. By contrast, when expressing a complex task using LTL, each component of the LTL property systematically structures the complex mission task into low-level, achievable task "modules" without requiring any supervision. The LTL property essentially acts as a high-level unsupervised guide for the agent, whereas the low-level planning is handled by a (deep) RL architecture. To demonstrate this benefit, we showcase our approach on the Atari 2600 game "Montezuma's Revenge", which is known as an exceptionally hard problem for RL.

Full instructions on reproducing all the case studies in this paper and how to import LCRL into any Python project are provided on the LCRL GitHub page:

`www.github.com/grockious/lcrl` `(@8b4e474)`

2. Overview

We give a brief overview of our approach. As is standard in existing methods, we convert the LTL property into an automaton, namely a finite-state machine [9]. However, LTL-to-automaton translation may result in a non-deterministic model, over which policy synthesis for MDPs is not semantically meaningful. A standard solution to this issue is to use Safra's construction to determinise the automaton. This is known to increase its size dramatically [120, 108]. An alternative solution is to convert the given LTL formula directly into a Deterministic Rabin Automaton (DRA). Such a conversion results, in the worst case, in automata that are doubly exponential in the size of the original LTL formula [4].

By contrast, we propose to express the given LTL property as a Limit-Deterministic Generalised Büchi Automaton (LDGBA) [126]. This construction results in a (singly) exponential-sized automaton for LTL\GU[1] and the result is nearly the same size as a DRA for the rest of LTL. We show that this succinctness improves the convergence speed and sample efficiency of the learning algorithm significantly. However, the translation of LTL into LDGBAs introduces non-trivial problems into the learning algorithm, which we address in this work. An additional benefit of our approach is that LDGBAs are semantically easier than DRAs owing to their acceptance conditions, which makes policy synthesis algorithms much simpler to implement [127, 137]. We emphasise that there exist a variety of algorithms for constructing limit-deterministic Büchi automata from LTL, but not all of resulting automata can be employed for quantitative model checking and probabilistic certification [80]. More importantly, the unique succinctness of the LDGBA construction [126] used in our work is due to its "generalised Büchi" accepting condition, and other, non-generalised constructions inevitably result in larger automata.

Once the LDGBA is generated from the given LTL property, we construct

[1] LTL\GU is a fragment of linear temporal logic with the restriction that no until operator occurs in the scope of an always operator.

on-the-fly[2] a product between the MDP and the resulting LDGBA. The generalised Büchi accepting condition then gives rise to a reward function that is synchronous with the state-action pairs of the MDP. Using this algorithmic reward-shaping procedure, an RL scheme is able to learn an optimal policy that returns the maximum expected reward, and that equivalently satisfies the given LTL specification with maximal probability.

Finally, the application of the LDGBA also allows the RL agent to generate episodes that are informative, particularly for the case of non-Markovian tasks: we show that this can reduce the sample complexity of RL architectures.

3. Background

We consider a universal RL setup, consisting of an agent interacting with an unknown environment, modelled as a general MDP.

3.1. Markov Decision Processes

Definition 1 (General MDP [14, 39]). The tuple $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ is a general MDP over a set of continuous states $\mathcal{S} = \mathbb{R}^n$, where $\mathcal{A} = \mathbb{R}^{n'}$ is a set of continuous actions, and $\mathcal{S}_0 \subset \mathcal{S}$ is the MDP initial set of states. An initial state s_0 is then randomly chosen from $\mathcal{S}_0$. $P : \mathcal{B}(\mathcal{S}) \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a Borel-measurable conditional transition kernel which assigns to any pair of state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ a probability measure $P(\cdot|s, a)$ on the Borel space $(\mathcal{S}, \mathcal{B}(\mathcal{S}))$, where $\mathcal{B}(\mathcal{S})$ is the set of all Borel sets on $\mathcal{S}$. $\mathcal{AP}$ is a finite set of atomic propositions and a labelling function $L : \mathcal{S} \to 2^{\mathcal{AP}}$ assigns to each state $s \in \mathcal{S}$ a set of atomic propositions $L(s) \subseteq 2^{\mathcal{AP}}$. ⌟

A finite-state/action MDP is a special case of general MDPs in which $|\mathcal{S}| < \infty$, $|\mathcal{A}| < \infty$, and $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function. The transition function P reduces to a transition probability matrix.

A variable $R(s, a) \sim \Upsilon(\cdot|s, a) \in \mathcal{P}(\mathbb{R}^+)$ can be defined over the MDP $\mathfrak{M}$, representing the immediate reward obtained when action a is taken at a given state s, where $\mathcal{P}(\mathbb{R}^+)$ is the set of probability distributions on subsets of $\mathbb{R}^+$, and Υ is the reward distribution. A realisation of R at time step n is denoted as r_n.

Definition 2 (Stationary Deterministic Policy). A stationary (randomised) policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a mapping from any state $s \in \mathcal{S}$ to a probability distribution over actions. A policy π assigns to any state $s \in \mathcal{S}$ a probability measure $\pi(\cdot|s)$ on the Borel space $(\mathcal{A}, \mathcal{B}(\mathcal{A}))$. A deterministic policy is a degenerate case of a randomised policy which outputs a single action at a given state, that is $\forall s \in \mathcal{S},\ \exists a \in \mathcal{A},\ \pi(s, a) = 1$. ⌟

Definition 3 (Expected Infinite-Horizon Discounted Return). For any policy π on an MDP $\mathfrak{M}$, and given a reward function R, the expected discounted

[2] "On-the-fly" means that the algorithm tracks (or executes) the state of an underlying structure (or a function) without explicitly building the entire structure.

reward return at state s is defined as [132]:

$$U^{\pi}_{\mathfrak{M}}(s) = \mathbb{E}^{\pi}[\sum_{n=0}^{\infty} \gamma^n \; r_n | s_0 = s], \quad s_n \sim P(\cdot|s_{n-1}, a_{n-1}), \; a_n \sim \pi(\cdot|s_n), \tag{1}$$

where $\mathbb{E}^{\pi}[\cdot]$ denotes the expected value given that the agent follows policy π from state s, $\gamma \in [0, 1)$ ($\gamma \in [0, 1]$ when episodic[3]) is a discount factor, and $s_0, a_0, s_1, a_1 ...$ is the sequence of states generated by policy π, initialised at $s_0 = s$. We will drop the subscript $\mathfrak{M}$ when it is clear from the context. ⌟

Note that the discount factor γ is a hyper-parameter that has to be tuned. In particular, there is standard work in RL with state-dependent discount factors that are shown to preserve convergence and optimality guarantees, by means of ensuring the contractivity of corresponding operators required to update value functions or policies [109, 153, 148, 103]. A possible tuning strategy for our setup would allow the discount factor be a function of the state:

$$\gamma(s) = \begin{cases} \eta & \text{if } R(s, a) > 0, \\ 1 & \text{otherwise,} \end{cases} \tag{2}$$

where $0 < \eta < 1$ is a constant. Hence, (1) would reduce to

$$U^{\pi}(s) = \mathbb{E}^{\pi}[\sum_{n=0}^{\infty} \gamma(s_n)^{N(s_n)} \; r_n) | s_0 = s], \tag{3}$$

where $N(s_n)$ is the number of times a positive reward has been observed at state s_n and $0 < \gamma(s) \leq 1$ [103].[4]

Definition 4 (Optimal Policy). An optimal policy π^* is defined as follows:

$$\pi^*(s) = \arg\sup_{\pi \in \mathcal{D}} \; U^{\pi}(s),$$

where $\mathcal{D}$ is the set of stationary deterministic policies over the state space $\mathcal{S}$ defined below. ⌟

Definition 5 (Semi-Deterministic Policy). We call a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ semi-deterministic if for some states, say $s \in \bar{S} \subset \mathcal{S}$, the policy π is a uniform distribution over a set of actions $\Lambda(s) \subseteq \mathcal{A}$, i.e., $\pi(s) = \mathit{unif}(\Lambda(s))$. In this work, whenever we talk about semi-deterministic policies, we assume that at any state $s \in \mathcal{S}$ the set $\Lambda(s)$ is the set of actions whose expected return is $U^{\pi^*}(s)$. In many works this specific interpretation of semi-determinism is simply referred to as policy determinism. However, a fully-deterministic policy is a special case of semi-deterministic policies where $\Lambda(s)$ is a singleton for all states $s \in \mathcal{S}$. ⌟

[3] An *episodic* RL algorithm consists of several reset trajectories, at each of which the agent re-starts from the initial state of the MDP.

[4] There are alternatives to this tuning strategy. For instance, [19] proposes state-dependent tuning where the reward function also depends on the discount factor, while in this work the reward function is independent of the discounting scheme.

An MDP $\mathfrak{M}$ is said to be solved if the agent discovers an optimal policy $\pi^* : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximises the expected return. We show later in the paper that synthesising a policy whose traces satisfy an LTL specification with maximum probability on MDP $\mathfrak{M}$ can be reduced to finding an optimal semi-deterministic policy that maximises an expected return on an extended MDP $\mathfrak{M}'$. In the following we review the syntax and semantics of LTL.

3.2. Linear Temporal Logic Properties

LTL is a rich task specification language that allows to express a wide range of properties (e.g., temporal, sequential, conditional). In this work, we use LTL specifications to formally and automatically shape a reward function which, as we discuss later, would otherwise be tedious to express and to achieve by conventional reward shaping methods.

Definition 6 (Path). In an MDP $\mathfrak{M}$, an infinite path ρ starting at s_0 is a sequence of states $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ such that every transition $s_i \xrightarrow{a_i} s_{i+1}$ is possible in $\mathfrak{M}$, i.e., s_{i+1} belongs to the smallest Borel set B such that $P(B|s_i, a_i) = 1$ (or in a finite-state MDP, s_{i+1} is such that $P(s_{i+1}|s_i, a_i) > 0$). We might also denote ρ as $s_0..$ to emphasise that ρ starts from s_0. ⌟

Given a path ρ, the i-th state of ρ is denoted by $\rho[i]$, where $\rho[i] = s_i$. Furthermore, the i-th suffix of ρ is $\rho[i..]$ where $\rho[i..] = s_i \xrightarrow{a_i} s_{i+1} \xrightarrow{a_{i+1}} s_{i+2} \xrightarrow{a_{i+2}} s_{i+3} \xrightarrow{a_{i+3}} \ldots$. The language of LTL formulae over a given set of atomic propositions $\mathcal{AP}$ is syntactically defined as [111]

$$\varphi := \mathit{true} \mid \alpha \in \mathcal{AP} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi \ \mathrm{U} \ \varphi,$$

where the operators $\bigcirc$ and U are called "next" and "until", respectively. The semantics of LTL formulae, as interpreted over MDPs, is discussed in the following.

Definition 7 (LTL Semantics). For an LTL formula φ and for a path ρ in an MDP $\mathfrak{M}$, the satisfaction relation $\rho \models \varphi$ is defined as [111, 9]

$$\begin{aligned}
&\rho \models \alpha \in \mathcal{AP} \Leftrightarrow \alpha \in L(\rho[0]),\\
&\rho \models \varphi_1 \wedge \varphi_2 \Leftrightarrow \rho \models \varphi_1 \wedge \rho \models \varphi_2,\\
&\rho \models \neg\varphi \Leftrightarrow \rho \not\models \varphi,\\
&\rho \models \bigcirc\varphi \Leftrightarrow \rho[1..] \models \varphi,\\
&\rho \models \varphi_1 \mathrm{U} \varphi_2 \Leftrightarrow \exists j \in \mathbb{N}_0 \text{ s.t. } \rho[j..] \models \varphi_2 \ \wedge \ \forall i,\ 0 \leq i < j,\ \rho[i..] \models \varphi_1.
\end{aligned}$$

⌟

The operator $\bigcirc$ (again, read as "next") requires φ to be satisfied starting from the next-state suffix of the path ρ. The operator U ("until") is satisfied over ρ if φ_1 continuously holds until φ_2 becomes *true*. By means of the until operator we are furthermore able to define two temporal modalities: (1) eventually, $\Diamond\varphi = \mathit{true}\ \mathrm{U}\ \varphi$; and (2) always, $\Box\varphi = \neg\Diamond\neg\varphi$. The intuition for $\Diamond\varphi$ is that φ has to become *true* at some finite point in the future, whereas $\Box\varphi$ means that φ has

to remain *true* forever. An LTL formula φ over $\mathcal{AP}$ specifies the following set of words:

$$Words(\varphi) = \{\sigma \in (2^{\mathcal{AP}})^{\omega} \text{ s.t. } \sigma \models \varphi\}. \tag{4}$$

Definition 8 (Safety Fragment of LTL). Let us define an unsafe prefix of a set of words, e.g., $Words(\cdot)$, as a finite word $\sigma_{finite} \in (2^{\mathcal{AP}})^*$ such that all infinite extensions, i.e., $\sigma_{finite}(2^{\mathcal{AP}})^{\omega}$, are not in $Words(\cdot)$. The safety fragment of LTL includes those formulae whose violating words has an unsafe prefix. ⌟

Definition 9 (Probability of Satisfying an LTL Formula). Starting from any state s and following a stationary semi-deterministic policy π, we denote the probability of satisfying formula φ as

$$Pr(s..^{\pi} \models \varphi),$$

where $s..^{\pi}$ denotes the collection of all paths starting from state s, generated under policy π. The maximum probability of satisfaction is also defined as:

$$Pr_{\max}(s_0 \models \varphi) = \sup_{\pi \in \mathcal{D}} Pr(s_0..^{\pi} \models \varphi).$$

⌟

Using an LTL formula we can now specify a set of constraints (i.e., requirements, or specifications) over the traces of the MDP. Once a policy π is selected, it dictates which action has to be taken at each state of the MDP $\mathfrak{M}$, hence reducing the MDP to a Markov chain denoted by $\mathfrak{M}^{\pi}$. For an LTL formula φ, an alternative method to express the set $Words(\varphi)$ in (4) is to employ a Limit-Deterministic Generalised Büchi automaton (LDGBA) [126]. We first define a Generalised Büchi Automaton (GBA), then we formally introduce the LDGBA [126].

Definition 10 (Generalised Büchi Automaton). A GBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is a structure where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $\Sigma = 2^{\mathcal{AP}}$ is a finite alphabet, $\mathcal{F} = \{F_1, ..., F_f\}$ is the set of accepting conditions, where $F_j \subset \mathcal{Q}, 1 \leq j \leq f$, and $\Delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$ is a transition relation. ⌟

Let Σ^{ω} be the set of all infinite words over Σ. An infinite word $\sigma \in \Sigma^{\omega}$ is accepted by a GBA $\mathfrak{A}$ if there exists an infinite run $\theta \in \mathcal{Q}^{\omega}$ starting from q_0 where $\theta[i+1] \in \Delta(\theta[i], \sigma[i]),\ i \geq 0$ and for each $F_j \in \mathcal{F}$

$$inf(\theta) \cap F_j \neq \emptyset, \tag{5}$$

where $inf(\theta)$ is the set of states that are visited infinitely often by the run θ.

Definition 11 (LDGBA). A GBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is limit-deterministic if $\mathcal{Q}$ can be partitioned into two disjoint sets $\mathcal{Q} = \mathcal{Q}_N \cup \mathcal{Q}_D$, such that [126]:

- $\mathcal{Q}_D$ is an invariant set: $\Delta(q, \alpha) \subset \mathcal{Q}_D$ and $|\Delta(q, \alpha)| = 1$ for every state $q \in \mathcal{Q}_D$ and for every $\alpha \in \Sigma$,

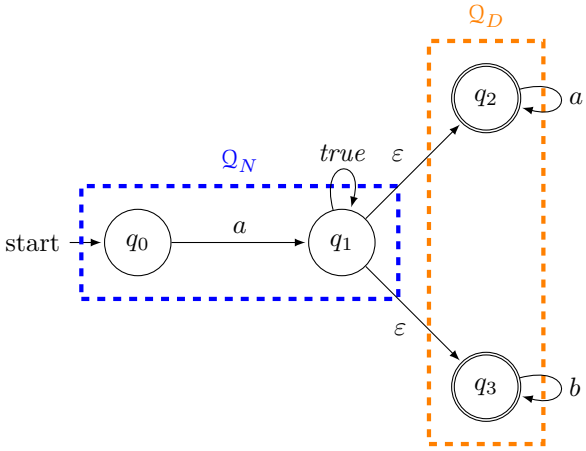


Figure 2: LDGBA for the formula $a \wedge \bigcirc(\Diamond\Box a \vee \Diamond\Box b)$.

- for every $F_j \in \mathcal{F}$, $F_j \subset \mathcal{Q}_D$,
- $q_0 \in \mathcal{Q}_N$, and all the transitions from $\mathcal{Q}_N$ to $\mathcal{Q}_D$ are non-deterministic ε-transitions.

Unlike a standard transition in a GBA, which requires a non-empty set of labels (see function Δ in Definition 10), in an ε-transition this set can be empty, which allows the automaton to change its state without reading any atomic proposition. ⌟

Intuitively, an LDGBA is a GBA that has two partitions: initial ($\mathcal{Q}_N$) and accepting ($\mathcal{Q}_D$). The accepting part includes all the accepting states and has deterministic transitions. As an example, Fig. 2 shows the LDGBA constructed for the formula $\varphi = a \wedge \bigcirc(\Diamond\Box a \vee \Diamond\Box b)$.

Remark 1. *The LTL-to-LDGBA algorithm used in this paper was proposed in [126]. It results in an automaton with two parts (initial $\mathcal{Q}_N$ and accepting $\mathcal{Q}_D$), both of which use deterministic transitions. Additionally, there are non-deterministic ε-transitions between them. According to Definition 11, the discussed structure is still an LDGBA (the determinism in the initial part is stronger than that required in the LDGBA definition). An ε-transition allows an automaton to change its state without reading an input symbol. In practice, during an episode of LCRL algorithm, whenever the agent reaches the boundary of $\mathcal{Q}_N$, e.g., state q_1 in Fig. 2, the ε-transitions between $\mathcal{Q}_N$ and $\mathcal{Q}_D$ reflect the agent's choices to move to $\mathcal{Q}_D$. The agent is free to choose any of these transitions as they do not require the agent to read any atomic proposition. This is later clarified further in Definition 13.* ⌟

Definition 12 (Non-accepting Sink Component). A non-accepting sink component of the LDGBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is a directed graph induced by a set of states $Q \subset \mathcal{Q}$ such that (1) the graph is strongly connected; (2) it does not

include all accepting sets $F_k,\ k = 1, ..., f$; and (3) there exist no other strongly connected set $Q' \subset \mathcal{Q},\ Q' \neq Q$, such that $Q \subset Q'$. We denote the union of all non-accepting sink components of $\mathfrak{A}$ as $\mathbb{N}$. ⌟

In the following we formally define the problem and discuss our proposed architecture Logically-Constrained Reinforcement Learning (LCRL).

4. Logically-Constrained Reinforcement Learning

We are interested in synthesising a policy (or policies) for an unknown (black-box) MDP via RL, such that the induced Markov chain satisfies a given LTL property with maximum probability.

Assumption 1. *In this paper, we assume that the MDP $\mathfrak{M}$ is fully unknown, and the learning agent has no prior knowledge about the transition kernel P.* ⌟

In the following, in order to explain the core concepts of the algorithm and for ease of exposition, let us abstractly assume for now that the MDP graph and the associated transition probabilities are known. Later these assumptions are entirely removed, and we stress that the algorithm can be run model-free. LCRL thus targets the issue of "verified learning" at its core, namely the model-free learning-based synthesis of policies that abide by a given LTL requirement. Furthermore, in order to handle the general case of non-ergodic MDPs, LCRL consists of several resets, at each of which the agent is forced to re-start from the initial state of the MDP: each reset defines an episode, as such the algorithm is known as "episodic RL".

Problem 1. *Given an MDP $\mathfrak{M}$ and an LTL specification φ, we wish to find an optimal policy $\pi^* \in \mathcal{D}$ such that the probability of satisfying the specification from any state is maximised, i.e., $\pi^* = \arg\sup_{\pi \in \mathcal{D}} Pr(s^\pi \models \varphi),\ \forall s \in \mathcal{S}$. Furthermore, we would like to find the maximum probability of satisfaction $Pr_{\max}(s \models \varphi)$.* ⌟

In order to tackle Problem 1, we first relate the MDP model and the LDGBA constructed from the given LTL formula, by "synchronising" the two. This new structure is, firstly, a good fit for RL and, secondly, it generates a model that satisfies the given LTL property.

Definition 13 (Product MDP). Given an MDP $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ and an LDGBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ with $\Sigma = 2^{\mathcal{AP}}$, the product MDP is defined as $(\mathfrak{M} \otimes \mathfrak{A}) = \mathfrak{M}_\mathfrak{A} = (\mathcal{S}^\otimes, \mathcal{A}^\otimes, s_0^\otimes, P^\otimes, \mathcal{AP}^\otimes, L^\otimes, \mathcal{F}^\otimes)$, where $\mathcal{S}^\otimes = \mathcal{S} \times \mathcal{Q}$, $\mathcal{A}^\otimes = \mathcal{A}$, $s_0^\otimes = (s_0, q_0)$, $\mathcal{AP}^\otimes = \mathcal{Q}$, $L^\otimes : \mathcal{S}^\otimes \to 2^\mathcal{Q}$ such that $L^\otimes(s, q) = q$ and $\mathcal{F}^\otimes \subseteq \mathcal{S}^\otimes$ is the set of accepting states $\mathcal{F}^\otimes = \{F_1^\otimes, ..., F_f^\otimes\}$, where $F_j^\otimes = \mathcal{S} \times F_j$. The transition kernel $P^\otimes$ is such that given the current state (s_i, q_i) and action a, the new state is (s_j, q_j), where $s_j \sim P(\cdot|s_i, a)$ and $q_j \in \Delta(q_i, L(s_j))$. When the MDP $\mathfrak{M}$ has a finite state space, then $P^\otimes : \mathcal{S}^\otimes \times \mathcal{A} \times \mathcal{S}^\otimes \to [0, 1]$ is the transition probability function, such that $(s_i \xrightarrow{a} s_j) \wedge (q_i \xrightarrow{L(s_j)} q_j) \Rightarrow P^\otimes((s_i, q_i), a, (s_j, q_j)) = P(s_i, a, s_j)$. Furthermore, in order to handle ε-transitions we make the following modifications to the above definition of product MDP:

- for every potential ε-transition to some state $q \in \mathcal{Q}$ we add a corresponding action ε_q in the product:

$$\mathcal{A}^{\otimes} = \mathcal{A} \cup \{\varepsilon_q, q \in \mathcal{Q}\}.$$

- the transition probabilities corresponding to ε-transitions are given by

$$P^{\otimes}((s_i, q_i), \varepsilon_q, (s_j, q_j)) = \begin{cases} 1 & \text{if } s_i = s_j,\ q_i \xrightarrow{\varepsilon_q} q_j = q, \\ 0 & \text{otherwise.} \end{cases}$$

⌟

Recall that an ε-transition between $\mathcal{Q}_N$ and $\mathcal{Q}_D$ corresponds in practical terms to a "guess" on reaching $\mathcal{Q}_D$. The intuition behind the above modification is that, during an exploration episode, once the automaton state reaches the edge of $\mathcal{Q}_N$ and an ε-transition is necessary, the product automaton treats this ε-transition as an extra action that can be taken in the MDP. Accordingly, during the RL exploration, if after an ε-transition the associated labels in the accepting set of the automaton cannot be read or the accepting states cannot be visited, then the guess is deemed wrong, and the exploration in RL is stopped.

Remark 2. *In order to clearly explain the role of different components in the proposed approach, we have employed model-dependent notions, such as transition probabilities and the product MDP. However, we emphasise again that the proposed approach can run "model-free", and as such that it does not depend on these components. In particular, as per Definition 11, the LDGBA is composed of two disjoint sets of states $\mathcal{Q}_D$ (which is invariant) and $\mathcal{Q}_N$, where the accepting states belong to the set $\mathcal{Q}_D$. Since all transitions are deterministic within $\mathcal{Q}_N$ and $\mathcal{Q}_D$, the automaton transitions can be executed simply by reading the labels, which makes the agent aware of the automaton state without explicitly constructing the product MDP. We will later define a reward function "on-the-fly", emphasising that the agent does not need to know the model structure or transition probabilities.* ⌟

Note that LTL is a temporal language and satisfying an LTL property requires a policy that is possibly non-Markovian and has an embedded memory [28, 29]. By constructing the product MDP we add an extra dimension to the state space of the original MDP, namely the states of the automaton representing the LTL formula. The role of the added dimension is to track LTL satisfaction and, hence, to synchronise the current state of the MDP with the state of the automaton: this essentially converts the non-Markovian LTL policy synthesis problem over the original MDP to a Markovian one over the product MDP.

In the following we briefly explain how a non-Markovian task can be broken down into simple composable Markovian sub-tasks (or modules). Each state of the automaton in the product MDP (Definition 13) is a "task divider" and each transition between these states is a "sub-task". For example consider a sequential task of visit a and then b and finally c, i.e.,

$$\Diamond(a \wedge \Diamond(b \wedge \Diamond c)).$$

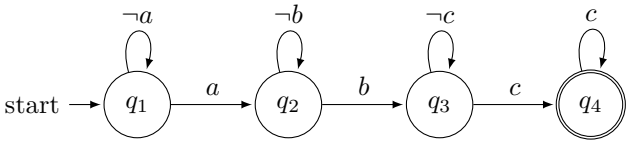


Figure 3: LDGBA for a sequential mission task.

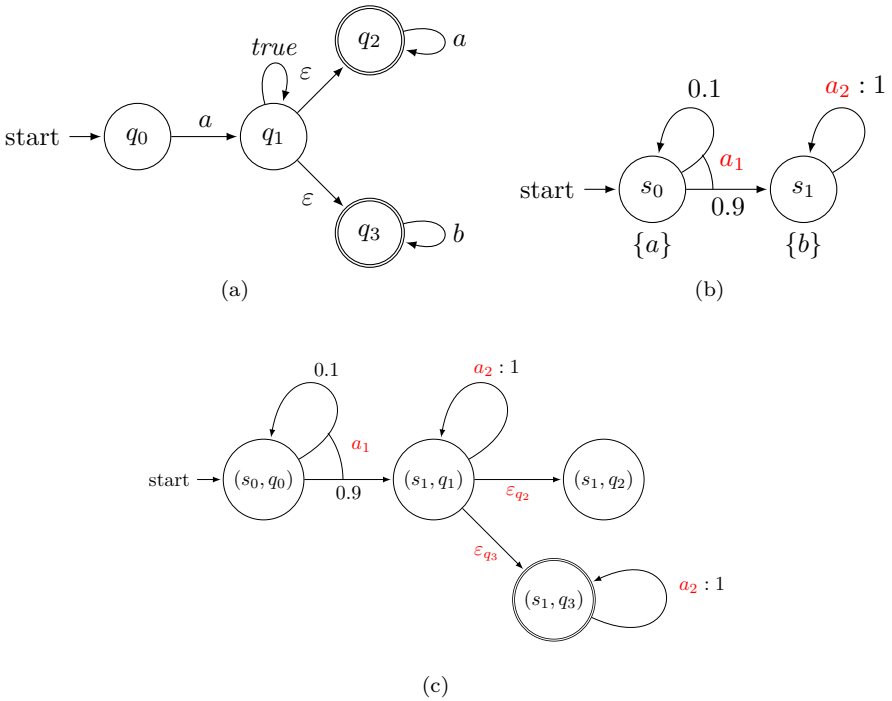


Figure 4: Example of product MDP: (a) the LDGBA from Fig. 2 and (b) an MDP; (c) the product of the MDP and the LDGBA, as per Definition 13. We rely on the observation [127] that it is sufficient to take ε-transitions only from states in the max-end components of the product of $\mathfrak{M}$ and the initial partition of the LDGBA $\mathfrak{Q}_N$. Hence no ε-transitions have to be produced in the initial state of $\mathfrak{M}_{\mathfrak{A}}$.

The corresponding automaton for this LTL task is given in Fig. 3. The entire task is modularised into three sub-tasks, i.e., reaching a, b, and then c, and each automaton state acts as a divider. For each automaton state q_i, RL needs to focus only on the outgoing edges of q_i. For instance, at q_2 in Fig. 3, RL only needs to find a policy whose traces satisfy the sub-formula $\Diamond b$. Fig. 4 illustrates the construction of a product MDP with the generated LDGBA in Fig. 2.

The product MDP (e.g., Fig. 4) provides a structure that allows us to shape a reward function for an RL algorithm, by leveraging the accepting condition of the LDGBA. Such a reward function thus clearly relates to the satisfaction of the given LTL formula. Before introducing such a reward assignment, we need to

define the ensuing function. Recall that a generalised Büchi automaton accepts words that visit its accepting sets infinitely often. The role of the following function is to track and output the subset of accepting sets that need to be visited at any given time. Namely, during the learning process, we would like to know precisely the set of labels that ought to be read (possibly once more), so that by such repeated visitations the specified LTL task is eventually satisfied.

Definition 14 (Accepting Frontier Function). Let $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$, be an LDGBA where $\mathcal{F} = \{F_1, ..., F_f\}$ is the set of accepting conditions, and $F_j \subset \mathcal{Q}, 1 \leq j \leq f$. Define the function $Acc : \mathcal{Q} \times \mathcal{F} \to 2^{\mathcal{Q}}$ as the accepting frontier function, which executes the following operation over a given set $\mathbb{F} \subset \mathcal{F}$ for every $F_j \in \mathcal{F}$:

$$Acc(q, \mathbb{F}) = \begin{cases} \mathbb{F} \backslash F_j & (q \in F_j) \wedge (\mathbb{F} \neq \{F_j\}), \\ \mathcal{F} \backslash F_j & (q \in F_j) \wedge (\mathbb{F} = \{F_j\}) \wedge (\mathcal{F} \backslash F_j \neq \emptyset), \\ \mathbb{F} & \text{otherwise.} \end{cases}$$

Once state $q \in F_j$ and set $\mathbb{F}$ are fed to function Acc, it outputs a set containing the elements of $\mathbb{F}$ minus F_j. However, if $\mathbb{F} = F_j$, then the output is the family of all accepting sets of the LDGBA minus the set F_j. Finally, if the state q is not an accepting state, then the output of Acc is $\mathbb{F}$. In short, the accepting frontier function excludes from $\mathbb{F}$ the accepting set that is currently visited, unless it is the only remaining accepting set. Otherwise, the output of $Acc(q, \mathbb{F})$ is $\mathbb{F}$ itself. What remains in $\mathbb{F}$ are those accepting sets that still need to be visited in order to attain the generalised Büchi accepting condition, as per Definition 10. ⌟

As discussed before, the product MDP encompasses the transition relations of the original MDP and the structure of the Büchi automaton, and it inherits characteristics of both. Thus, a proper reward function leads the RL agent to find a policy that is optimal, in the sense that it satisfies the LTL property φ with maximal probability. We introduce an on-the-fly reward function that fits the model-free RL architecture: when an agent observes the current state $s^{\otimes}$, implements action a and observes the subsequent state $s^{\otimes\prime}$, the agent is given a scalar reward, as follows:

$$R(s^{\otimes}, a) = \begin{cases} r_p & \text{if } q' \in \mathbb{A},\ s^{\otimes\prime} = (s', q'), \\ r_n & \text{otherwise.} \end{cases} \tag{6}$$

Here, $r_p > 0$ is a positive reward and $r_n = 0$ is a neutral reward. A positive reward is assigned to the agent when it takes an action that leads to a state with a label in $\mathbb{A}$. The set $\mathbb{A}$ is called the accepting frontier set, is initialised as the family of sets $\mathbb{A} = \{F_k\}_{k=1}^{f} = \mathcal{F}$, and is updated by the following rule every time after the reward function is evaluated:

$$\mathbb{A} \leftarrow Acc(q', \mathbb{A}).$$

The set $\mathbb{A}$ always contains the set of accepting states that ought to be visited at any given time: in this sense the reward function is "synchronous" with the

accepting condition set by the LDGBA. Thus, the agent is guided by the above reward assignment to visit those states and once all the sets $F_k,\ k = 1, ..., f$, are visited, the accepting frontier $\mathbb{A}$ is reset. As a consequence, as the agent continuously explores, it is guided to visit all the accepting sets infinitely often and is rewarded towards the satisfaction of the corresponding LTL property. Considering the syntactic tree of the LTL property, by visiting each accepting set F_k infinitely often, the agent satisfies the corresponding sub-formula of the LTL formula. This means that, by guiding the agent to visit all the accepting sets, we are essentially guiding the agent to move upwards in the syntactic tree towards the satisfaction of the entire LTL property. We elaborate on the issue of partial satisfaction of the LTL property in the supplementary material (Appendix B).

The reward structure depends on parameters $r_p = M + y \times m \times rand(s^{\otimes})$ and $r_n = y \times m \times rand(s^{\otimes})$. The parameter $y \in \{0, 1\}$ is a constant, $0 < m \ll M$ are arbitrary positive values, and $rand : \mathcal{S}^{\otimes} \to (0, 1)$ is a function that generates a random number in $(0, 1)$ for each state $s^{\otimes}$ each time R is being evaluated. The role of the function *rand* is to resolve possible symmetry issues[5] when neural nets are used for approximating the Q-function (namely, when the MDP state space is continuous). Also, note that parameter y acts as a switch to bypass the effect of the *rand* function on R when no neural net is used. Thus, this switch is active $y = 1$ when the MDP state space is continuous, and disabled in other cases $y = 0$.

Remark 3. *As our implementation is model-free, note that when running the proposed algorithm there is no need to "explicitly build" the product MDP and to store all its states and transitions in memory. The automaton transitions can be executed on-the-fly as the agent reads the labels of the MDP states. Namely, the agent can track the automaton state by just looking at the trace that has been read so far. The agent only needs to store the current state of the automaton and observe the label at each step to check whether the automaton state has changed or not.* ⌟

In the following, we further elaborate on how the automaton state tracks the evolution of the accepting frontier set $\mathbb{A}$.

Proposition 1. *Given an LTL formula φ and its associated LDGBA $\mathfrak{A} = (Q, q_0, \Sigma, \mathcal{F}, \Delta)$, the set members of $\mathbb{A}$ only depend on the current state of the automaton and not on the sequence of automaton states that have been already visited. (proof in the supplementary materials)*

Proposition 1 allows us to reason about the evolution of the accepting frontier set $\mathbb{A}$ throughout the learning process. Specifically, the accepting sets are removed from $\mathbb{A}$ upon visiting accepting sets in Q_D until all accepting sets are visited at least once. This resets the accepting frontier set $\mathbb{A}$ (Definition 14), and thus

[5] If all weights in a feedforward net start with equal values and if the solution requires unequal weights be learnt, the neural network might not proceed to learn. Identical weights within the same hidden layer induce symmetries, which the neural net must break in order to generalise, reduce redundancies on the weights, and optimise the loss function [70].

the agent receives a positive reward infinitely often for visiting accepting sets as specified in (6). Given this reward structure, we show in the following that the optimal policy generated by an RL scheme maximises, in the limit, the probability of satisfying the LTL property.

Theorem 1. *Let φ be the given LTL property and $\mathfrak{M}_\mathfrak{A}$ be the product MDP constructed by synchronising the MDP $\mathfrak{M}$ and the LDGBA $\mathfrak{A}$ expressing φ. There exists a discount factor that is close enough to 1 under which an optimal Markov policy on $\mathfrak{M}_\mathfrak{A}$ that maximises the expected return over the reward in (6) also maximises the probability of satisfying φ. This optimal Markov policy induces a finite-memory policy on the MDP $\mathfrak{M}$. (proof in the supplementary materials)*

Remark 4. *Note that the projection of policy π^* onto the state space of the original MDP $\mathfrak{M}$ yields a finite memory policy $\pi^*_\mathfrak{M}$. Thus, if the generated traces under π^* maximise the LTL satisfaction probability, so do the traces under $\pi^*_\mathfrak{M}$.* ⌟

Remark 5. *The optimality of the policies generated using (deep) neural-network-based approaches depends on a number of factors, such as the network structure, number of hidden layers, and activation functions. Specifically, convergence guarantees for such methods to a true optimal policy are not well developed and quantification of the sub-optimality of the policy generated by these methods is out of the scope of this work.* ⌟

An interesting extension of Theorem 1 is the following corollary.

Corollary 1 (Maximum Probability of Satisfaction)**.** *From Definition 9, for a discounting factor close enough to 1 the maximum probability of satisfaction at any state $s^\otimes$ can be determined from the LCRL value function as*

$$Pr_{\max}(s^\otimes \models \varphi) = \frac{1-\eta}{r_p}\, U^{\pi^*}(s^\otimes).$$

(proof in the supplementary materials)

Remark 6. *As mentioned in Definition 5, semi-deterministic and deterministic policies are used interchangeably in a number of works. However, in general, at any state $s^\otimes$, there might be multiple actions $\Lambda(s^\otimes)$ whose expected return is $U^{\pi^*}(s^\otimes)$. An optimal action is then uniformly selected from $\Lambda(s^\otimes)$ as per Definition 5. Let us assume for the sake of simplicity that the action space is finite and we expand the definition of the expected utility from Definition 3:*

$$Pr_{\max}(s^\otimes \models \varphi) = \frac{1-\eta}{r_p} \times \sum_{a' \in \Lambda(s^\otimes)} \frac{1}{|\Lambda(s^\otimes)|} \mathbb{E}^{\pi^*}\Big[\sum_{n=0}^{\infty} \gamma(s^\otimes_n)^{N(s^\otimes_n)}\, r_n \Big| s^\otimes_0 = s^\otimes, a_0 = a'\Big], \tag{7}$$

where $1/|\Lambda(s^\otimes)|$ is the probability of either of actions in $\Lambda(s^\otimes)$ to be selected. Since all the actions in $\Lambda(s^\otimes)$ have the same expected return $U^{\pi^*}(s^\otimes)$, the

conditional expectation on the RHS of (7) is independent of a'. Thus, it can be factored out from the summation:

$$Pr_{\max}(s^{\otimes} \models \varphi) = \frac{1-\eta}{r_p}\ U^{\pi^*}(s^{\otimes}) \sum_{a' \in \Lambda(s^{\otimes})} \frac{1}{|\Lambda(s^{\otimes})|}.$$

Note that $\sum_{a' \in \Lambda(s^{\otimes})} 1/|\Lambda(s^{\otimes})| = 1$, which means that π^* maximises the satisfaction probability even when the optimal policy is semi-deterministic. In case when the action space is not finite, the summation in (7) is an integral. ⌟

Theorem 2. *Let n be the length of the formula φ, i.e., the size of its syntactic tree or the number of $\bigcirc$ and* U *operators used in φ as per Definition 7. Then for any LTL formula in the LTL$\setminus$GU fragment, there exist an LDGBA with size of only $2^{\mathcal{O}(n)}$ while an equivalent DRA has a size of $2^{2^{\mathcal{O}(n)}}$ [127, 79].*

It is easy to find examples similar to the LTL$\setminus$GU fragment [126] in addition to efficient procedures for LDGBA complementation [17]. As per Theorem 2, LDGBAs are much more succinct than DRAs, and hence in LCRL, the space over which the learning is performed (i.e., $\mathfrak{M}_{\mathfrak{A}}$) is $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)2^{\mathcal{O}(n)}$. Given that in principle an RL agent has to visit all the state-action pairs of the learning space to converge [132], this exponential reduction in the size of $\mathfrak{M}_{\mathfrak{A}}$ significantly improves convergence speed and sample efficiency. Note that, in the worst case, LTL to LDGBA translation is doubly-exponential, which is just as expensive as LTL to DRA, and therefore, LDBGAs are in practice smaller than DRAs.

As a concluding remark, notice that LCRL is a general policy synthesis architecture that is adaptable to many off-the-shelf model-free RL algorithms. The LCRL RL Engine in Fig. 1 is a good fit for a broad variety of RL schemes that conform with the required state and action space cardinality and dimension. Within the LCRL architecture, the MDP and LDGBA states are synchronised, resulting in an on-the-fly product MDP. In the following sections we then discuss the applicability of LCRL, and we present case studies to demonstrate the ease of use, and scalability of the scheme.

5. Logically-Constrained Tabular RL

Recall that the simplest case in Definition 1 is when the MDP state space and action space are both finite. This case, however, covers a significant number of control applications. In this section we discuss and show LCRL, a model-free RL architecture with discounted reward, can be efficiently employed for LTL policy synthesis and quantitative model checking in finite MDPs.

Q-learning (QL) is the most extensively used RL algorithm for synthesising optimal policies in finite-state MDPs [132]. In this section, the LCRL RL Engine in Fig. 1 is QL: we run QL over the product MDP $\mathfrak{M}_{\mathfrak{A}}$ with the reward shaping proposed in (6), where we have set $y = 0$. In order to also handle non-ergodic MDPs, we propose to employ a variant of standard QL that consists of several resets, at each of which the agent is forced to re-start from its initial state s_0.

Each reset defines an episode, and hence the algorithm is called "episodic QL". However, for the sake of brevity, we omit the term "episodic" in the rest of the paper and we use the term Logically-Constrained QL (LCQL).

For each state $s^{\otimes} \in \mathcal{S}^{\otimes}$ and for any available action $a \in \mathcal{A}^{\otimes}$, QL assigns a quantitative value $Q : \mathcal{S}^{\otimes} \times \mathcal{A}^{\otimes} \to \mathbb{R}$, which is initialised with an arbitrary and finite value over all state-action pairs. As the agent starts receiving rewards and learning, the Q-function is updated by the following rule for taking action a at state $s^{\otimes}$:

$$Q(s^{\otimes}, a) \leftarrow (1-\mu)Q(s^{\otimes}, a) + \mu\big[R(s^{\otimes}, a) + \gamma(s^{\otimes}) \max_{a' \in \mathcal{A}^{\otimes}} (Q(s'^{\otimes}, a'))\big], \tag{8}$$

where $Q(s^{\otimes}, a)$ is the Q-value corresponding to state-action $(s^{\otimes}, a)$, $0 < \mu \leq 1$ is called learning rate or step size, $R(s^{\otimes}, a)$ is the reward obtained for performing action a in state s, γ is the discount factor, and $s'^{\otimes}$ is the state obtained after performing action a. The Q-function for the rest of the state-action pairs remains unchanged.

Under mild assumptions over the learning rate [15, 147], for finite-state and -action spaces QL converges to a unique limit. This unique limit is the expected discounted reward by taking action a at state s, and following the optimal policy afterwards. Let us call this limit Q^*. Once QL converges, an optimal policy $\pi^* : \mathcal{S}^{\otimes} \to \mathcal{A}^{\otimes}$ can be generated by selecting the action that yields the highest Q^*, i.e.,

$$\pi^*(s^{\otimes}) \in \arg\max_{a \in \mathcal{A}} \; Q^*(s^{\otimes}, a).$$

Here π^* corresponds to the optimal policy that can be generated via DP. This means that when QL converges, we have

$$Q^*(s^{\otimes}, a) = R(s^{\otimes}, a) + \gamma(s^{\otimes}) \sum_{s'^{\otimes} \in \mathcal{S}^{\otimes}} P(s^{\otimes}, a, s'^{\otimes}) U^{\pi^*}(s'^{\otimes}),$$

where $s'^{\otimes}$ is the agent new state after choosing action a at state $s^{\otimes}$ such that $P(s'^{\otimes} | s^{\otimes}, a) > 0$.

6. Logically-Constrained Neural Fitted Q-iteration

LCQL is focused on problems in which the set of states of the MDP and the set of possible actions are both finite. Nonetheless, many interesting real-world problems require actions to be taken in response to high-dimensional or real-valued states [37]. We can collect a number of samples and only then apply an approximation function that is constructed via regression over the set of samples. The approximation function essentially replaces the conventional LCQL state-action-reward look-up table by generalising over the state space of the MDP. In this section, we extend the LCRL architecture to a model-free RL algorithm based on Neural Fitted Q-iteration (NFQ), which can synthesise an optimal policy for an LTL property when the given MDP has a continuous state space. We replace the QL algorithm in LCRL RL Engine in Fig. 1 by NFQ to showcase the

Algorithm 1: Logically-Constrained QL

input : LTL specification, *it_threshold*, γ, μ
output : π^*

initialize $Q : \mathcal{S}^{\otimes} \times \mathcal{A}^{\otimes} \to \mathbb{R}_0^+$
convert the desired LTL property to LDGBA $\mathfrak{A}$
initialize $\mathbb{A}$
initialize *episode-number* $:= 0$
initialize *iteration-number* $:= 0$
while Q *is not converged* **do**
  episode-number $++$
  $s^{\otimes} = (s_0, q_0)$
  while $(q \notin \mathbb{N} : \ s^{\otimes} = (s, q))$ & (*iteration-number* $<$ *it_threshold*) **do**
    iteration-number $++$
    choose $a_* = \pi(s^{\otimes}) \in \arg\max_{a \in \mathcal{A}} Q(s^{\otimes}, a)$ # $\epsilon-$greedy or softmax are applicable
    move to $s_*^{\otimes} = (s_*, q_*)$ by a_*
    receive the reward $R(s^{\otimes}, a_*)$
    $\mathbb{A} \leftarrow Acc(q_*, \mathbb{A})$
    $Q(s^{\otimes}, a_*) \leftarrow$ $Q(s^{\otimes}, a_*) + \mu[R(s^{\otimes}, a_*) - Q(s^{\otimes}, a_*) + \gamma(s^{\otimes}) \max_{a'}(Q(s_*^{\otimes}, a'))]$
    $s^{\otimes} = s_*^{\otimes}$
  end
end

flexibility of the LCRL architecture. We call this algorithm Logically-Constrained NFQ (LCNFQ) and we show that the proposed architecture is efficient and is compatible with RL algorithms that are core of recent developments in the community. We have studied a number of alternative RL-based approaches to LCNFQ in [60], and LCNFQ easily outperformed the competitors.

NFQ is an algorithm that employs feedforward neural networks [71] to approximate the Q-function, namely to efficiently generalise or interpolate it over the entire state space, exploiting a finite set of experience samples. This set is called experience replay. Instead of the conventional QL update rule in (8), NFQ introduces a loss function that measures the error between the current Q-values $Q(s, a)$ and their target value $R(s, a) + \gamma \max_{a'} Q(s', a')$, namely

$$L = [Q(s, a) - R(s, a) + \gamma \max_{a'} Q(s', a')]^2. \tag{9}$$

In LCNFQ, the experience replay method is adapted to the product MDP structure, over which we let the agent explore the MDP and reinitialise it when a positive reward is received or when no positive reward is received after a given number *th* of iterations. The parameter *th* is set manually according to the state space of the MDP, allowing the agent to explore the MDP while keeping the size of the sample set limited. All the traces that are gathered within episodes, i.e.,

Algorithm 2: Logically-Constrained NFQ

input : the set of experience samples $\mathcal{E}$
output : approximated Q-function

initialise all neural nets B_{q_i} with (s_0, q_i, a) as the input and r_n as the output where $a \in \mathcal{A}$ is a random action
repeat
  for $q_i = |\mathcal{Q}|$ ***to*** 1 **do**
    $\mathcal{P}_{q_i} = \{(input_l, target_l),\ l = 1, ..., |\mathcal{E}_{q_i}|)\}$
      $input_l = ({s_l}^{\otimes}, a_l)$
      $target_l = R({s_l}^{\otimes}, a_l) + \gamma \max_{a'} Q({s_l}^{\otimes\prime}, a')$
      where $({s_l}^{\otimes}, a_l, {s_l}^{\otimes\prime}, R({s_l}^{\otimes}, a_l), q_i) \in \mathcal{E}_{q_i}$
    $B_{q_i} \leftarrow \text{Rprop}(\mathcal{P}_{q_i})$
  end
until *end of trial*

experiences, are stored in the form of $(s^{\otimes}, a, s^{\otimes\prime}, R(s^{\otimes}, a), q)$, where $s^{\otimes} = (s, q)$ is the current state in the product MDP, a is the selected action, $s^{\otimes\prime} = (s', q')$ is the subsequent state, and $R(s^{\otimes}, a)$ is the reward gained as in (6) with $y = 1$ in r_p. The set of past experiences is called the sample set $\mathcal{E}$.

Once the exploration phase is completed and the sample set is created, learning is performed over the sample set. In the learning phase, we propose a hybrid architecture of n separate feedforward neural nets, each with one hidden layer, where $n = |\mathcal{Q}|$ and $\mathcal{Q}$ is the finite cardinality of the automaton $\mathfrak{A}$[6]. Each neural net is associated with a state in the LDGBA and for each automaton state $q_i \in \mathcal{Q}$ the associated neural net is called $B_{q_i} : \mathcal{S}^{\otimes} \times \mathcal{A} \to \mathbb{R}$. Once the agent is at state $s^{\otimes} = (s, q_i)$, the neural net B_{q_i} is used for the local Q-function approximation. The set of neural nets acts as a global hybrid Q-function approximator $Q : \mathcal{S}^{\otimes} \times \mathcal{A} \to \mathbb{R}$. Note that the neural nets are not fully decoupled. For example, assume that by taking action a in state $s^{\otimes} = (s, q_i)$ the agent is moved to state $s^{\otimes\prime} = (s', q_j)$ where $q_i \neq q_j$. According to (9) the weights of B_{q_i} are updated such that $B_{q_i}(s^{\otimes}, a)$ has minimum possible error to $R(s^{\otimes}, a) + \gamma \max_{a'} B_{q_j}(s^{\otimes\prime}, a')$. Therefore, the value of $B_{q_j}(s^{\otimes\prime}, a')$ affects $B_{q_i}(s^{\otimes}, a)$.

Let $q_i \in \mathcal{Q}$ be a state in the LDGBA. Then define $\mathcal{E}_{q_i} := \{(\cdot, \cdot, \cdot, \cdot, x) \in \mathcal{E} | x = q_i\}$ as the set of experiences within $\mathcal{E}$ that are associated to state q_i, i.e., $\mathcal{E}_{q_i}$ is

[6]Different embeddings, such as the one-hot encoding [56] and the integer encoding, have been applied in order to approximate the global Q-function with a single feedforward net. However, we have observed poor performance since these encodings allow the network to assume relationships between automaton states that might not agree with the automaton structure. Clearly, this disrupts Q-function generalisation by assuming that some states in product MDP are closer to each other. Consequently, we have turned to the use of n separate neural nets, which work together in a hybrid fashion, meaning that the agent can switch between these neural nets as it jumps from one automaton state to another.

the projection of $\mathcal{E}$ onto q_i. Once the exploration phase is completed, each neural net B_{q_i} is trained based on the associated experience set $\mathcal{E}_{q_i}$. At each iteration of training, a pattern set $\mathcal{P}_{q_i}$ is generated based on the experience set $\mathcal{E}_{q_i}$:

$$\mathcal{P}_{q_i} = \{(input_l, target_l), l = 1, ..., |\mathcal{E}_{q_i}|)\},$$

where $input_l = ({s_l}^{\otimes}, a_l)$ and $target_l = R({s_l}^{\otimes}, a_l) + \gamma \max_{a'} Q({s_l}^{\otimes'}, a')$ such that $({s_l}^{\otimes}, a_l, {s_l}^{\otimes'}, R({s_l}^{\otimes}, a_l), q_i) \in \mathcal{E}_{q_i}$. In each epoch of LCNFQ (Algorithm 2), the pattern set $\mathcal{P}_{q_i}$ is used as the input-output set to train the neural net B_{q_i}. In order to update the weights in each neural net, we use Rprop [118] for its efficiency in batch learning [117]. The training schedule in the hybrid network starts from individual networks that are associated with accepting states of the automaton. The training sequence goes backward until it reaches the networks that are associated to the initial states. By doing so, we allow the Q-value to back-propagate through the connected networks. In Algorithm 2, without loss of generality we assume that the automaton states are ordered and hence the back-propagation starts from $q_i = |\mathcal{Q}|$. Despite the improvement in performance and applicability, LCNFQ cannot deal with the most general case of MDPs, i.e., continuous state-action MDPs. Thus, in the following we extend LCRL towards an online learning scheme that efficiently handles continuous state-action MDPs.

7. Modular Deep Actor-critic Learning

The previous section introduced LCNFQ, a model-free neural-fitted RL scheme. LCNFQ exploits the positive effects of generalisation in feedforward nets. Feedforward nets are efficient in predicting Q-values for state-action pairs that have not been visited by interpolating between available data. This means that the learning algorithm requires less experience and the learning process is thus data efficient. However, LCNFQ is an offline learning algorithm, i.e., experience gathering and learning happens separately. Furthermore, when dealing with the most general case of MDPs, i.e., uncountably infinite-state and infinite-action, LCNFQ is of limited use. An obvious approach to adapt LCNFQ to continuous action domains is to discretise the action space. However, this has many limitations such as loss of dynamics accuracy, and most importantly the curse of dimensionality: the number of actions exponentially increases with the number of degrees of freedom in the MDP.

Policy gradient methods, on the other hand, are online schemes that are widely used in RL for MDPs with continuous action spaces. The general idea is to represent the policy by a parametric probability distribution $\pi(\cdot|s, \theta^{\pi})$ and then adjusting the policy parameters θ^{π} in the direction of the greatest cumulative reward. This policy can of course be deterministic, but there is a crucial difference between the stochastic and deterministic policy gradients [107]. From a practical point of view, stochastic policy gradients require more experience samples. In this work we focus on deterministic policies, as they are sufficient for most control problems and because deterministic policy gradients are more efficient in terms of sample complexity.

The actor-critic architecture is a widely used method based on the policy gradient [132, 155], which consists of two interacting components: an actor and a critic. The actor is the parametric policy $\pi(\cdot|s,\theta^\pi)$ (or $\pi(s|\theta^\pi)$ when the policy is deterministic), and the critic is an action-value function $Q(s,a)$ that guides the updates of parameters θ^π in the direction of the greatest cumulative reward. The Deterministic Policy Gradient (DPG) algorithm [129] introduces a parameterised deterministic function $\pi(s|\theta^\pi)$ as the actor to represent the current policy by deterministically mapping states to actions, where θ^π are the function approximation parameters for the actor function. A parameterised action-value function $Q(s,a|\theta^Q)$ is the critic and is learned as described next.

Assume that at time step n the agent is at state s_n, takes action a_n, and receives a scalar reward $R(s_n,a_n)$ as in (6) with $y=1$ in r_p. The action-value function update is then approximated by parameterising Q using a parameter set θ^Q, i.e., $Q(s_n,a_n|\theta^Q)$, and by minimising the following loss function:

$$L(\theta^Q)=\mathbb{E}^{\pi}_{s_n\sim\rho^\beta}[(Q(s_n,a_n|\theta^Q)-\Xi_n)^2], \tag{10}$$

where ρ^β is the probability distribution of state visits over $\mathcal{S}$ under any given arbitrary stochastic policy β, and $\Xi_n=R(s_n,a_n)+\gamma Q(s_{n+1},a_{n+1}|\theta^Q)$. The parameters of the actor $\pi(s|\theta^\pi)$ are updated by applying the chain rule to the expected return with respect to the actor parameters, which is approximated as follows [129]:

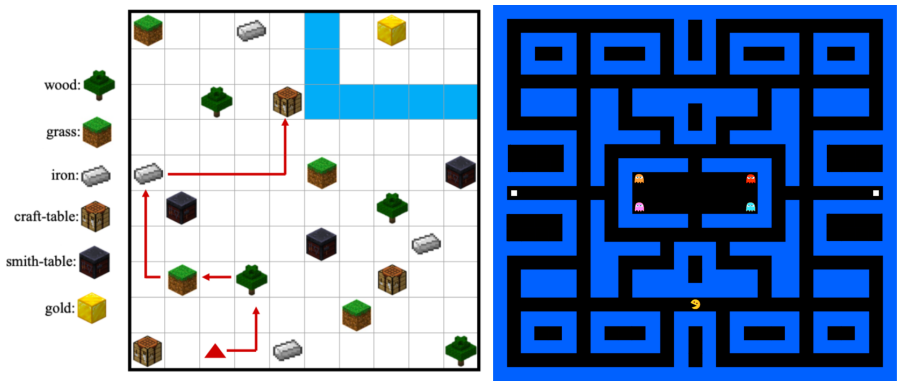
$$\begin{aligned}\nabla_{\theta^\pi}U^\pi(s_n)&\approx\mathbb{E}_{s_n\sim p^\beta}[\nabla_{\theta^\pi}Q(s,a|\theta^Q)|_{s=s_n,a=\pi(s_n|\theta^\pi)}]\\&=\mathbb{E}_{s_n\sim p^\beta}[\nabla_a Q(s,a|\theta^Q)|_{s=s_n,a=\pi(s_n)}\nabla_{\theta^\pi}\pi(s|\theta^\pi)|_{s=s_n}].\end{aligned}$$

The results in [129] show that this is a policy gradient, and therefore we can apply a policy gradient algorithm on the deterministic policy. Deep DPG (DDPG) further extends DPG by employing a deep neural network as function approximator and updating the network parameters via a "soft update" method, similar to [101], and is thoroughly explained later.

Given an LTL task and its LDGBA $\mathfrak{A}=(\mathcal{Q},q_0,\Sigma,\mathcal{F},\Delta)$, we propose a modular architecture of $n=|\mathcal{Q}|$ separate actor, actor-target, critic and critic-target neural networks, along with separate replay buffers. For each automaton state q_i, an actor function $\mu_{q_i}(s|\theta^{\mu_{q_i}})$ represents the current policy, where $\theta^{\mu_{q_i}}$ is the vector of parameters of the function approximation for the actor. The critic $Q_{q_i}(s,a|\theta^{Q_{q_i}})$ is learned based on (10).

The set of neural nets acts as a global modular actor-critic deep RL architecture, which allows the agent to jump from one sub-task to another by just switching between the set of neural nets. The proposed modular DDPG algorithm is detailed in Algorithm 3. Each actor-critic network set in this algorithm is associated with its own replay buffer $\mathcal{E}_{q_i}$, where $q_i\in\mathcal{Q}$ (lines 4 and 12).

At each time-step, actor and critic are updated by sampling a mini-batch of size $\mathcal{M}$ uniformly from $\mathcal{E}_{q_i}$. We only train the actor-critic network set corresponding to the current automaton state, as experience samples on the current automaton state have little influence on other actor-critic networks (lines 12–17).



(a) (b)

Figure 5: (a) sample run by LCRL policy learnt for Task 3. (b) `pacman-lrg`—the square on the left is labelled as food 1 (f_1) and the one on the right as food 2 (f_2), the state of being caught by a ghost is labelled as (g) and the rest of the state space is neutral (n).

Further, directly implementing the update of the critic parameters as in (10) is known to be potentially unstable, and as a result the Q-update (line 14) is prone to divergence [100]. Hence, instead of directly copying the weights, the standard DDPG [95] uses "soft" target updates to improve learning stability. The target networks Q' and μ' are time-delayed copies of the original actor and critic networks that slowly track the learned networks, Q and μ. These target actor and critic networks are used within the algorithm to gather evidence (line 13) and subsequently to update the actor and critic networks. In our algorithm, for each automaton state q_i we make a copy of the actor and the critic network, denoted by $\mu'_{q_i}(s|\theta^{\mu'_{q_i}})$ and $Q'_{q_i}(s,a|\theta^{Q'_{q_i}})$, respectively. The weights of both target networks are then updated by $\theta' = \tau\theta + (1-\tau)\theta'$ with a rate of $\tau < 1$ (line 18).

8. Experimental Results

We have tested LCRL on a number of planning experiments that require policy synthesis for a given temporal specification, both when the state and action spaces are finite, and when they are continuous. Despite dealing with an unknown MDP, we have observed that LCRL results are comparable to those of model-checking methods whenever available, as will be discussed in detail in this section. All the experiments are run on a standard PC, with an Intel Core i5 CPU at 2.5 GHz $\times$ 4 and with 20 GB of RAM.

The experiments are listed in Table 1. For each MDP, the state space cardinality $|\mathcal{S}|$ and the action space cardinality $|\mathcal{A}|$ are given, and for each LTL property the number of automaton states $|\mathcal{Q}|$ is reported. For each MDP-automaton product pair, Table 1 presents the maximum probability of satisfaction of the LTL objective at the initial state when using the strategy synthesised by LCRL.

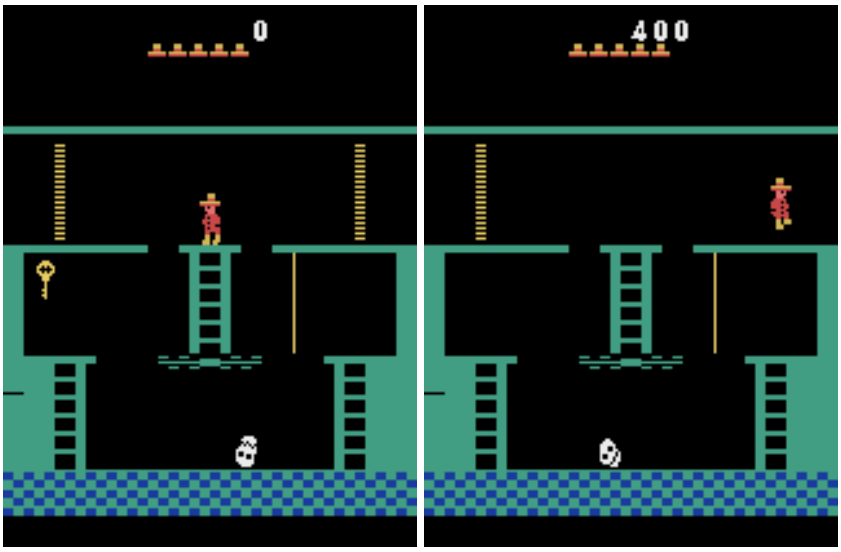
Algorithm 3: Modular DDPG

input : LTL mission task φ, black-box model
output : actor and critic networks

convert the LTL property φ to LDGBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$
randomly initialise $|\mathcal{Q}|$ actors $\mu_i(s|\theta^{\mu_i})$ and critic $Q_i(s, a|\theta^{Q_i})$ networks with weights θ^{μ_i} and θ^{Q_i}, for each $q_i \in \mathcal{Q}$, and all state-action pairs (s, a)
initialise $|\mathcal{Q}|$ corresponding target networks μ'_i and Q'_i with weights $\theta^{\mu'_i} = \theta^{\mu_i}$, $\theta^{Q'_i} = \theta^{Q_i}$
initialise $|\mathcal{Q}|$ replay buffers $\mathcal{E}_i$
repeat
  initialise $|\mathcal{Q}|$ random processes $\mathfrak{N}_i$
  initialise state $s_1^{\otimes} = (s_0, q_0)$
  for $t = 1$ ***to*** *max_iteration_number* **do**
    choose action $a_t = \mu_{q_t}(s_t|\theta^{\mu_{q_t}}) + \mathfrak{N}_{q_t}$
    observe reward R_t and the new state (s_{t+1}, q_{t+1})
    store $((s_t, q_t), a_t, R_t, (s_{t+1}, q_{t+1}))$ in $\mathcal{E}_{q_t}$
    sample a random mini-batch of $\mathcal{M}$ transitions $((s_i, q_i), a_i, R_i, (s_{i+1}, q_{i+1}))$ from $\mathcal{E}_{q_t}$
    set $\Xi_i = R_i + \gamma Q'_{q_{i+1}}(s_{i+1}, \mu'_{q_{i+1}}(s_{i+1}|\theta^{\mu'_{q_{i+1}}})|\theta^{Q'_{q_{i+1}}})$
    update critic Q_{q_t} and $\theta^{Q_{q_t}}$ by minimising the loss: $L = 1/|\mathcal{M}| \sum_i (\Xi_i - Q_{q_t}(s_i, a_i|\theta^{Q_{q_t}}))^2$
    update the actor policy μ_{q_t} and $\theta^{\mu_{q_t}}$ by maximising the sampled policy gradient:
    $\nabla_{\theta^{\mu_{q_t}}} U^{\mu_{q_t}} \approx 1/|\mathcal{M}| \sum_i [\nabla_a Q_{q_t}(s, a|\theta^{Q_{q_t}})|_{s=s_i, a=\mu_{q_t}(s_i|\theta^{\mu_{q_t}})}$
    $\nabla_{\theta^{\mu_{q_t}}} \mu_{q_t}(s|\theta^{\mu_{q_t}})|_{s=s_i}]$
    update the target networks: $\theta^{Q'_{q_t}} \leftarrow \tau\theta^{Q_{q_t}} + (1-\tau)\theta^{Q'_{q_t}}$
    $\theta^{\mu'_{q_t}} \leftarrow \tau\theta^{\mu^{q_t}} + (1-\tau)\theta^{\mu'_{q_t}}$
  end
until *end of trial*

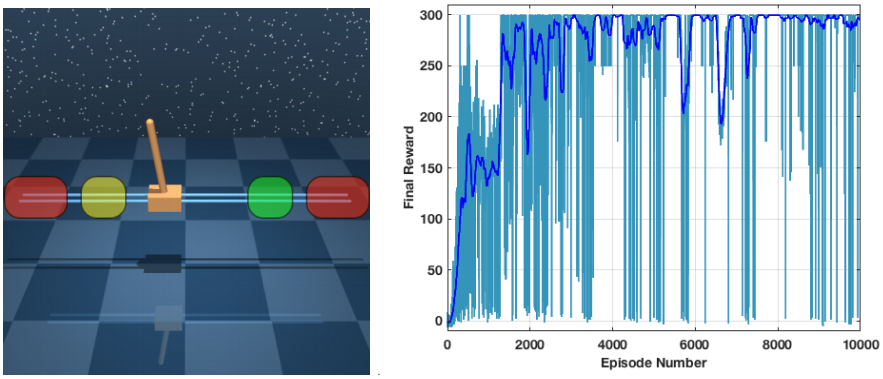
As a reference, we also compute the maximum satisfaction probability using the PRISM model checker [83] and report it in the table, whenever computationally feasible (the model checker has scalability limits, as expected, that prevent completion on two of the benchmarks). The remaining columns give the values of the hyper-parameters. All the reported results are the averages of ten learning trials done with LCRL.

The `minecraft` environment, adopted from [7], requires solving challenging low-level control tasks (`minecraft-tX`), and features many sequential high-level goals. For instance, in `minecraft-t3` (Fig. 5.a) the agent has to collect three items sequentially, and to reach a final checkpoint, which is encoded as the



(a) (b)

Figure 6: (a) `montezuma` (Montezuma's Revenge) initial frame (b) agent successfully unlocks a door.



(a) (b)

Figure 7: (a) `cart-pole` experiment [134]; (b) learning curves (dark blue) obtained averaging over 10 randomly initialised experiment, shaded areas (light blue) represent envelopes of 10 generated learning curves.

following LTL formula:

$$\Diamond(wood \wedge \Diamond(grass \wedge \Diamond(iron \wedge \Diamond(craft\_table))))$$

The `mars-rover` problems are realistic benchmarks taken from [60, 63], and

Table 1: Learning results with LCRL. MDP state and action space cardinalities are $|\mathcal{S}|$ and $|\mathcal{A}|$, the number of automaton states in LDBA is denoted by $|\mathcal{Q}|$, the optimal action value function in the initial state is denoted by "LCRL $\max_a Q(s_0, a)$", which represents the LCRL estimation of the maximum satisfaction probability. For each experiment, the reported result includes the mean and the standard error of ten learning trials with LCRL. This probability is also calculated by the PRISM model checker [83], whenever the MDP model can be processed by PRISM, and accordingly it is reported in the column "`max sat. prob. at` s_0". The closer "LCRL $\max_a Q(s_0, a)$" and "`max sat. prob. at` s_0" the better. Note that for continuous state-action MDPs the maximum satisfaction probability cannot be precisely computed by model checking tools, unless abstraction approximation techniques (outside of the scope of this work, cf. [89]) are applied, hence "n/a". Furthermore, if the MDP state (or action) space is large enough, e.g. `pacman`, the process (either learning or model-checking) times out, i.e. "t/o". The column "`episode_num`" presents the episode number in which LCRL converged, using LDBA and also DRA as the underlying automaton. The rest of the columns provide the values of the hyper-parameters across the different benchmarks.

experiment	MDP $\lvert\mathcal{S}\rvert, \lvert\mathcal{A}\rvert$	LDBA $\lvert\mathcal{Q}\rvert$	LCRL $\max_a Q(s_0, a)$	max sat. prob. at s_0	alg.	episode_ num (LDBA/DRA)	iteration_ num_max	discount_ factor*	learning_ rate†	wall_clock time★(min)
minecraft-t1	100, 5	3	0.991 ± 0.009	1	'ql'	500/1000	4000	0.95	0.9	0.1
minecraft-t2	100, 5	3	0.991 ± 0.009	1	'ql'	500/1000	4000	0.95	0.9	0.1
minecraft-t3	100, 5	5	0.993 ± 0.007	1	'ql'	1500/6000	4000	0.95	0.9	0.25
minecraft-t4	100, 5	3	0.991 ± 0.009	1	'ql'	500/1000	4000	0.95	0.9	0.1
minecraft-t5	100, 5	3	0.995 ± 0.005	1	'ql'	500/1000	4000	0.95	0.9	0.1
minecraft-t6	100, 5	4	0.995 ± 0.005	1	'ql'	1500/4200	4000	0.95	0.9	0.25
minecraft-t7	100, 5	5	0.993 ± 0.007	1	'ql'	1500/6000	4000	0.95	0.9	0.5
mars-rover-1	∞, 5	3	0.991 ± 0.002	n/a	'nfq'	50/170	3000	0.9	0.01	550
mars-rover-2	∞, 5	3	0.992 ± 0.006	n/a	'nfq'	50/150	3000	0.9	0.01	540
mars-rover-3	∞, ∞	3	n/a	n/a	'ddpg'	1000/3800	18000	0.99	0.05	14
mars-rover-4	∞, ∞	3	n/a	n/a	'ddpg'	1000/3000	18000	0.99	0.05	12
mars-rover-5	∞, ∞	13	n/a	n/a	'ddpg'	17e3/ t/0	25000	0.9	0.003	300
cart-pole	∞, ∞	4	n/a	n/a	'ddpg'	100/420	10000	0.99	0.02	1
montezuma	∞, 18	7	n/a	n/a	'dqn'	400e3/ t/o	150e3	0.99	0.00025	>4000
robot-surve	25, 4	3	0.994 ± 0.006	1	'ql'	500/1000	1000	0.95	0.9	0.1
slp-easy-sml	120, 4	2	0.974 ± 0.026	1	'ql'	300/500	1000	0.99	0.9	0.1
slp-easy-med	400, 4	2	0.990 ± 0.010	1	'ql'	1500/2700	1000	0.99	0.9	0.25
slp-easy-lrg	1600, 4	2	0.970 ± 0.030	1	'ql'	2000/3500	1000	0.99	0.9	2
slp-hard-sml	120, 4	5	0.947 ± 0.039	1	'ql'	500/1800	1000	0.99	0.9	1
slp-hard-med	400, 4	5	0.989 ± 0.010	1	'ql'	4000/9000	2100	0.99	0.9	5
slp-hard-lrg	1600, 4	5	0.980 ± 0.016	1	'ql'	6000/15000	3500	0.99	0.9	9
frozen-lake-1	120, 4	3	0.949 ± 0.050	0.9983	'ql'	400/800	2000	0.99	0.9	0.1
frozen-lake-2	400, 4	3	0.971 ± 0.024	0.9982	'ql'	2000/4100	2000	0.99	0.9	0.5
frozen-lake-3	1600, 4	3	0.969 ± 0.019	0.9720	'ql'	5000/9400	4000	0.99	0.9	1
frozen-lake-4	120, 4	6	0.846 ± 0.135	0.9728	'ql'	2000/9000	2000	0.99	0.9	1
frozen-lake-5	400, 4	6	0.735 ± 0.235	0.9722	'ql'	7000/ t/o	4000	0.99	0.9	2.5
frozen-lake-6	1600, 4	6	0.947 ± 0.011	0.9467	'ql'	5000/ t/o	5000	0.99	0.9	9
pacman-sml	729,000, 5	6	0.290 ± 0.035	t/o‡	'ql'	80e3/400e3	4000	0.95	0.9	1600
pacman-lrg	4,251,000, 5	6	0.282 ± 0.049	t/o‡	'ql'	180e3/ t/o	4000	0.95	0.9	3700

* coefficient η in (2) † learning rate μ ‡ timed out due to the state space size ★ on a machine running macOS 11.6.5 with Intel Core i5 CPU at 2.5 GHz and with 20 GB of RAM

the models feature uncountably infinite (continuous) state and action spaces. Fig. 8.a gives the constructed automaton for `mars-rover-5`. The `cart-pole` experiment (Fig. 7) is explained in [134, 63, 22], and the property of interest is expressed by the LTL formula

$$\square\lozenge y \wedge \square\lozenge g \wedge \square\neg u.$$

The pole starts upright, and the goal is to prevent the pendulum from falling over ($\square\neg u$), and to move the cart between the yellow (y) and green (g) regions ($\square\lozenge y \wedge \square\lozenge g$), while avoiding the red (unsafe) parts of the track ($\square\neg u$).

Atari 2600 Montezuma's Revenge `montezuma` is an infamously hard exploration problem. The task in `montezuma` (Fig 6.a) is to climb down to the floor from the top of the middle ladder, jump over the skull, fetch the key and move back up to open either doors. The example `robot-surve` is adopted from [119], and the task is to visit two regions (A and B) in sequence, while avoiding multiple

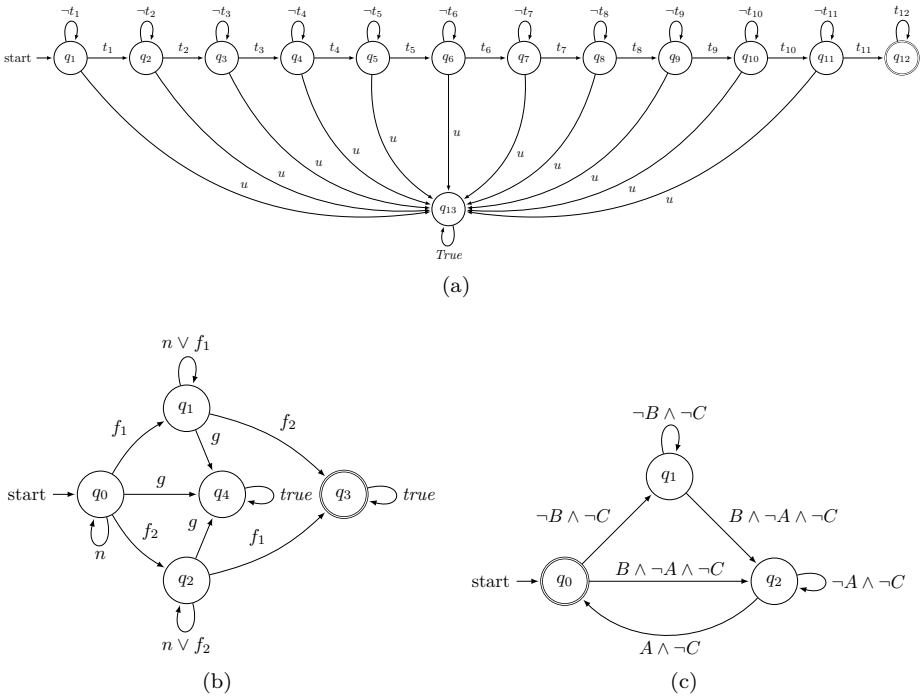


Figure 8: LDBA for the specification in (a) `mars-rover-5`; (b) `pacman-sml` and `pacman-lrg`; and (c) `robot-surve`.

obstacles (C) on the way: $\square\lozenge A \wedge \square\lozenge B \wedge \square\neg C$. The LDGBA expressing this LTL formula is presented in Fig. 8.c. Models `slp-easy` and `slp-hard` are inspired by the widely used noisy MDPs in [132, Chapter 6]: the goal in `slp-easy` is to reach a particular region of the MDP, whereas the goal in `slp-hard` is to visit four distinct regions sequentially in a given order. The `frozen-lake` benchmarks are similar: the first three are reachability problems, whereas the last three require sequential visits of four regions in the presence of unsafe regions as well. The `frozen-lake` MDPs are adopted from the OpenAI Gym [21].

Finally, `pacman-sml` and `pacman-lrg` are inspired by the well-known Atari game Pacman, and are initialised in a tricky configuration (`pacman-lrg` as in Fig. 5.b) likely to lead to getting caught: to win the game the agent has to collect all available tokens without being caught by moving ghosts. Formally, the agent is required to choose between one of the two available foods and then find the other one ($\lozenge[(f_1 \wedge \lozenge f_2) \vee (f_2 \wedge \lozenge f_1)]$) while avoiding the ghosts ($\square\neg g$). We feed a conjunction of these associations to the agent by using the following LTL formula$\lozenge[(f_1 \wedge \lozenge f_2) \vee (f_2 \wedge \lozenge f_1)] \wedge \square\neg g$. The LDGBA expressing this LTL formula is presented in Fig. 8.b. Standard QL failed to find a policy with satisfying traces. Similar to LCRL, the reward function in standard QL is positive if the agent manages to achieve the specified task, and zero otherwise. However, one major difference is that the state space in standard QL is not

enriched with the automaton, and hence any generated policy is memoryless.

9. Related Work

The goal of control (policy) synthesis in finite-state/action MDPs, under temporal logic specifications, has been considered in numerous works. In [150], the property of interest is expressed in LTL and converted to a DRA to synthesise a robust control policy. Specifically, a product MDP is constructed with the resulting DRA and a modified DP is applied to the product MDP, maximising the worst-case probability of satisfying the specification over all transition probabilities. However, [150] assumes that the MDP is known a-priori. This assumption is relaxed in [40] and transition probabilities in the given MDP model are considered to be unknown. Instead, a Probably Approximately Correct MDP (PAC MDP) is constructed and further multiplied by the logical property after conversion to a DRA. The overall goal is to calculate a finite-horizon T-step value function for each state such that the obtained value is within an error bound from the probability of satisfying the given LTL property.

The goal of maximising the probability of satisfying unbounded-time reachability properties, when the MDP transition probabilities are unknown, is investigated in [20]. The policy generation relies on approximate DP, which requires a mechanism to approximate these probabilities (much like the PAC MDP above), and the quality of the generated policy critically depends on the accuracy of this approximation, which might require a large number of simulation runs. Furthermore, the algorithm in [20] assumes prior knowledge about the smallest transition probability in the MDP. Using an LTL-to-DRA conversion, the algorithm given in [20] can be extended to the problem of control synthesis for LTL specifications, at the expense of a double-exponential blow-up of the obtained automaton.

Much in the same direction, if there exists a policy whose traces satisfy the property with probability one, [119] employs a learning-based approach to generate one such policy. As for [20], the algorithm in [119] hinges on approximating the transition probabilities, which affects precision and scalability of the approach.

Our work on model-free RL for LTL has been first introduced in [58]. Since then, growing research has been devoted to model-free RL with different kinds of automata [25], including limit-deterministic Büchi automata [54, 19], where different forms of reward schemes have been examined [92, 27, 98, 76, 75]. Specifically, [54] employs a limit-deterministic Büchi automata based on a Non-deterministic Büchi Automaton (NBA) that is structurally different than [126]'s LDGBA, used in this work. Bozkurt and others [19] extend upon the NBA-based automata, by proposing an interleaving reward and discounting scheme, and [22] employs policy gradient to tackle MDPs with high-dimensional state/action spaces.

There has been significant interest also in specifying tasks in RL via subfragments of LTL [138, 28, 29, 88] and other forms of temporal logic [77, 146, 10]. The problem of synthesising a policy that satisfies a temporal logic specification

and that at the same time optimises a performance criterion is considered in [93, 133, 35, 149]. In [82], scLTL is proposed for mission specifications, which results in Deterministic Finite Automata (DFAs). A product MDP is then constructed and a linear programming solver is used to find optimal policies. [66, 97] synthesise DFAs on-the-fly in the context of deep RL and inverse RL, to infer an scLTL property. Conversely, Mealy machines are inferred in [116] to represent a temporal non-Markovian task in Monte Carlo tree search. PCTL specifications are investigated in [84], where a linear optimisation solution is used to synthesise a control policy. In [105], an automated method is proposed to verify and repair the policies that are generated by RL with respect to a PCTL formula - the key engine runs by feeding the Markov chain induced by the policy to a probabilistic model checker. In [5], practical challenges of RL are addressed by letting the agent plan ahead in real time using constrained optimisation.

In [149], the problem is separated into two sub-problems: extracting a (maximally) permissive strategy for the agent and then quantifying the performance as a reward function and computing an optimal strategy for the agent within the operating envelope allowed by the permissive strategy. Similarly, [74] first computes safe, permissive strategies with respect to a reachability property. Then, under these constrained strategies, RL is applied to synthesise a policy that minimises an expected cost. The concept of shielding is employed in [3] to synthesise a policy that ensures that the agent remains safe during and after learning for a fully-deterministic reactive system. This approach is closely related to teacher-guided RL [135]. In order to express the temporal specification, [3] uses DFAs and then translates the problem into a safety game. The game is played between the environment and the agent, where in every state of the game the environment chooses an input, and then the agent selects an output. The game is won by the agent if only safe states are visited during the play. However, the generated policy always needs the shield to be online, as the shield maps every unsafe action to a safe action. The work in [73] extends [3] to probabilistic systems modelled as MDPs with adversarial uncontrollable agents. The general assumption in [73] is that the controllable agent acquires full observations over the MDP and the adversarial agent: unlike the proposed method in this work, the RL scheme used in [73] is model-based and requires the agent to first build a model of the stochastic MDP. The concept of a bounded-prescience shield is proposed in [48] for analysing and ensuring the safety of deep RL agents in Atari games. [41, 44, 42, 141] address safety-critical settings in the context of cyber-physical systems, where the agent has to deal with a heterogeneous set of models in model-based RL. [44] first generates a set of feasible models given an initial model and data on runs of the system. With such a set of feasible models, the agent has to learn how to safely identify which model is the most accurate one. [43] further employs differential dynamic logic [110], a first-order multimodal logic for specifying and proving properties of hybrid models.

Safe RL is an active area of research whose focus is on the efficient implementation of safety properties, and is mostly based on reward engineering [46, 12]. Our proposed method is related to work on safe RL, but cannot simply be reduced to it, due to its generality and to its inherent structural differences. By

focusing on the safety fragment of LTL, the proposed scheme does not require the reward function to be handcrafted. The reward function is automatically shaped by exploiting the structure of the LDGBA and its generalised Büchi acceptance condition. However, for the safety fragment of LTL the proposed automatic reward shaping can be seen as a way of "modifying the optimisation criterion," as in [46]. Additionally, we would like to emphasise that our work cannot be considered a Constrained MDP (CMDP) method, as the LTL satisfaction is encoded in the expected return itself, while in CMDP algorithms the original objective is separated from the constraint. In a nutshell, the proposed method inherits reward engineering aspects that are standard in safe RL, however at the same time it infuses notions from formal methods that allow guiding exploration and certifying its outcomes. Note that safe RL in general is not equivalent to ensuring agent safety during learning. Learning-while-being-safe by itself is a semantically-different research area that has attracted considerable attention recently, e.g., [112, 61, 12, 50, 140, 139, 26, 121, 99]

A relevant body of work had been done on both finite- and continuous-state-action MDPs, when the MDP model is fully known [89]. Probabilistic reachability over a finite horizon for hybrid continuous-state-action MDPs is investigated in [1], where a DP-based algorithm is employed to produce safe policies. DFAs have been employed in [136] to find an optimal policy for infinite-horizon probabilistic reachability problems. FAUST^2 [130], StocHy [31], and AMYTISS [87] deal with uncountable-state MDPs by generating a discrete-state abstraction based on the knowledge of the MDP model. Using probabilistic bi-simulation [53] showed that abstraction-based model checking can be effectively employed to generate control policies in continuous-state/action MDPs. Bounded LTL is proposed in [94] as the specification language, and a policy search method is used for synthesis. Automatic control approaches are also studied to deal with partially infeasible LTL specifications [24, 23].

Statistical Model Checking (SMC) techniques have also been studied for policy synthesis in MDPs, however they are not well suited to models that exhibit non-determinism. This is due to the fact that SMC techniques often rely on generation of random paths, which are not well-defined for an MDP with non-determinism [18, 91]. Some SMC approaches proposed to resolve the MDP non-determinism by using uniform distributions [36, 85] and others proposed to consider all possible strategies [86, 68] and produced policies that are close to an optimal one. Unlike RL, which improves its exploration policy during learning, a constant random policy is expected to waste time and computational resources to generate sample traces. Also, a trace is "only" used to reinforce each state-action pair visited by the associated path if the trace satisfies the property of interest. This is quite similar to Monte Carlo methods rather than RL or DP. For this reason, SMC methods are not expected to scale as well as RL. Further, sampling and checking of traces needs to be computationally feasible: SMC techniques are effective with finite-horizon LTL properties, as opposed to the focus of this work on infinite-horizon properties and full LTL. The efforts on statistical model-checking of unbounded properties is limited to a few specifications [154]. However, there have been some recent developments,

e.g., [68], that leverage RL to reduce the randomness in the policy.

This article summarises and extends material presented in earlier conference publications [22, 60, 61, 62, 63, 64, 65], whilst presenting it in a more comprehensive manner and with a unique set of experiments. More specifically, in this article we present and expand rigorous theoretical guarantees, upon which our earlier work [22, 64] was established. This includes new definitions, propositions, and theorems on automata and RL theory along with complexity analysis. Furthermore, we present new experiments that are significantly more complicated and sophisticated than those presented in [22, 64]—an example is the well-known Atari game Montezuma's Revenge which is still a hard problem for modern AI solutions. These new experimental results are a significant improvement over the results presented in [60, 61, 62, 63], and further support the claims on performance, correctness, and sample efficiency [57].

10. Conclusions

We have proposed LCRL, a method for guiding the training of an RL agent using a priori knowledge about the environment given in the form of an LTL property, expressed via an automaton known as LDBA. This additional knowledge, as we have observed in many experiments presented in this work, improves training drastically. We have shown that the policy synthesised using LCRL is guaranteed to maximise the probability of satisfying the LTL property: this architecture is shown to be working across many environments, whether with finite or continuous states and actions. This direct relationship between the expected return of the policy generated using LCRL and the maximum probability of satisfaction enables us to quantify the degree of safety of the generated policy for any given state.

Good avenues for future work are problems in which the specification is initially unknown and has to be discovered along the learning process [66]. Furthermore, there are relevant synergies between the use of LTL to guide an agent and LTL to restrict the exploration of the agent during training ("learning while staying safe") [61]. Multi-agent setups, in which a (heterogeneous) set of agents collaborate to satisfy an LTL formula is an interesting research direction with various applications [55].

Appendices

A. Proofs

Before proving Proposition 1, let us present the following definition.

Definition 15 (G-sub-formula). Given an LTL property φ and a set of G-sub-formulae $\mathcal{G}$[7] we define $\varphi[\mathcal{G}]$ the resulting formula when we substitute *true* for every G-sub-formula in $\mathcal{G}$ and $\neg true$ for other G-sub-formulae of φ. ⌟

Proposition 1. *Given an LTL formula φ and its associated LDGBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$, the set members of $\mathbb{A}$ only depend on the current state of the automaton and not on the sequence of automaton states that have been already visited.*

proof.
Let $\mathcal{G} = \{\Box\zeta_1, ..., \Box\zeta_f\}$ be the set of all G-sub-formulae of φ. Since elements of $\mathcal{G}$ are sub-formulae of φ we can assume an ordering over $\mathcal{G}$, so that if $\Box\zeta_i$ is a sub-formula of $\Box\zeta_j$, then $j > i$. In particular, $\Box\zeta_f$ is not a sub-formula any G-sub-formula. The accepting component of the LDGBA $\mathcal{Q}_D$ is a product of f of the DBAs $\{\mathbf{D}_1, ..., \mathbf{D}_f\}$ called G-monitors, such that each $\mathbf{D}_i = (\mathcal{Q}_i, q_{i0}, \Sigma, F_i, \delta_i)$ expresses $\Box\zeta_i[\mathcal{G}]$, where $\mathcal{Q}_i$ is the state space of the i-th G-monitor, $\Sigma = 2^{\mathcal{AP}}$, and $\delta_i : \mathcal{Q}_i \times \Sigma \to \mathcal{Q}_i$ [126]. Note that $\zeta_i[\mathcal{G}]$ has no G-sub-formula any more (Definition 15). The states of the G-monitor $\mathbf{D}_i$ are pairs of formulae where at each state the the G-monitor only checks if the run satisfies $\Box\zeta_i[\mathcal{G}]$ while putting the next G-sub-formula in the ordering of $\mathcal{G}$ on hold, assuming that it is *true*.

The product of G-monitor DBAs is a deterministic generalised Büchi automaton:

$$\mathbf{P_D} = (\mathcal{Q}_D, q_{D0}, \Sigma, \mathcal{F}, \delta)$$

where $\mathcal{Q}_D = \mathcal{Q}_1 \times ... \times \mathcal{Q}_f$, $\Sigma = 2^{\mathcal{AP}}$, $\mathcal{F} = \{F_1, ..., F_f\}$, and $\delta = \delta_1 \times ... \times \delta_f$.

As shown in [126], while a word w is being read by the accepting component of the LDGBA, the set of G-sub-formulae that hold is "monotonically" expanding. If $w \in \mathit{Words}(\varphi)$, then eventually all G-sub-formulae become *true*. Now, let the current state of the automaton be $q_D = (q_1, ..., q_i, ..., q_f)$ while the automaton is checking whether $\Box\zeta_i[\mathcal{G}]$ is satisfied or not, assuming that $\Box\zeta_{i+1}$ is already *true* (though needs to be checked later), while all G-monitors $\Box\zeta_j[\mathcal{G}]$, $1 \leq j \leq i-1$ have accepted w. At this point, the accepting frontier set is $\mathbb{A} = \{F_i, F_{i+1}, ..., F_f\}$. Reasoning by contradiction, assume that the automaton returns to q_D but $\mathbb{A} \neq \{F_i, F_{i+1}, ..., F_f\}$, then at least one accepting set F_j, $j > i$ has been removed from $\mathbb{A}$. This means that $\Box\zeta_j$ is a sub-formula of $\Box\zeta_i$, violating the ordering of check on $\mathcal{G}$. This is a contradiction with respect to the ordering of G-sub-formulae.

[7] A G-sub-formula is a sub-formula of φ of the form $\Box(\cdot)$.

Theorem 3. *Let φ be the given LTL property and $\mathfrak{M}_{\mathfrak{A}}$ be the product MDP constructed by synchronising the MDP $\mathfrak{M}$ and the LDGBA $\mathfrak{A}$ expressing φ. There exists a discount factor that is close enough to* 1 *under which an optimal Markov policy on $\mathfrak{M}_{\mathfrak{A}}$ that maximises the expected return over the reward in (6), also maximises the probability of satisfying φ. This optimal Markov policy induces a finite-memory policy on the MDP $\mathfrak{M}$.*

proof.
Assume that there exists a policy $\overline{\pi}$ that satisfies φ with maximum (non-zero) probability. Policy $\overline{\pi}$ induces a Markov chain $\mathfrak{M}_{\mathfrak{A}}^{\overline{\pi}}$ when it is applied over the MDP $\mathfrak{M}_{\mathfrak{A}}$. This Markov chain comprises a disjoint union between a set of transient states $\mathfrak{T}_{\overline{\pi}}$ and h sets of irreducible recurrent classes $\mathfrak{R}_{\overline{\pi}}^{i},\ i = 1, ..., h$ [39], namely:

$$\mathfrak{M}_{\mathfrak{A}}^{\overline{\pi}} = \mathfrak{T}_{\overline{\pi}} \sqcup \mathfrak{R}_{\overline{\pi}}^{1} \sqcup ... \sqcup \mathfrak{R}_{\overline{\pi}}^{h}.$$

From (5), policy $\overline{\pi}$ satisfies φ if and only if:

$$\exists \mathfrak{R}_{\overline{\pi}}^{i} \text{ s.t. } \forall j \in \{1, ..., f\},\ F_j^{\otimes} \cap \mathfrak{R}_{\overline{\pi}}^{i} \neq \emptyset. \tag{11}$$

The recurrent classes that satisfy (11) are called accepting. From the irreducibility of the recurrent class $\mathfrak{R}_{\overline{\pi}}^{i}$ we know that all the states in $\mathfrak{R}_{\overline{\pi}}^{i}$ communicate with each other thus, once a trace ends up in such set, all the accepting sets are going to be visited infinitely often. Therefore, from the definition of $\mathbb{A}$ and of the accepting frontier function (Definition 14), the agent receives a positive reward r_p ever after it has reached an accepting recurrent class $\mathfrak{R}_{\overline{\pi}}^{i}$.

There are two other possibilities concerning the remaining recurrent classes that are not accepting. A non-accepting recurrent class, name it $\mathfrak{R}_{\overline{\pi}}^{k}$, either

1. has no intersection with any accepting set $F_j^{\otimes}$, i.e.

$$\forall j \in \{1, ..., f\},\ F_j^{\otimes} \cap \mathfrak{R}_{\overline{\pi}}^{k} = \emptyset;$$

2. or has intersection with some of the accepting sets but not all of them, i.e.

$$\exists J \subset 2^{\{1,...,f\}} \setminus \{1, ..., f\} \text{ s.t. } \forall j \in J,\ F_j^{\otimes} \cap \mathfrak{R}_{\overline{\pi}}^{k} \neq \emptyset. \tag{12}$$

In the second case, the agent is able to visit some accepting sets but not all of them. This means that in the update rule of the frontier accepting set $\mathbb{A}$ in Definition 14, the case where $(q \in F_j) \wedge (\mathbb{A} = F_j)$ will never happen since there exist always at least one accepting set that has no intersection with $\mathfrak{R}_{\overline{\pi}}^{k}$. Therefore, after a limited number of times, no positive reward can be obtained, and the reinitialisation of $\mathbb{A}$ in Definition 14 is blocked.

Recall Definition 3, where the expected return for a state $s^{\otimes} \in \mathcal{S}^{\otimes}$ is defined as in (2) and (3):

$$U^{\overline{\pi}}(s^{\otimes}) = \mathbb{E}^{\overline{\pi}}[\sum_{n=0}^{\infty} \gamma(s_n^{\otimes})^{N(s_n^{\otimes})}\ r_n) | s_0^{\otimes} = s^{\otimes}],$$

In both cases, from (6), for any arbitrary $r_p > 0$ (and $r_n = 0$), there always exists a discounting coefficient η such that the expected return of a trace reaching $\mathfrak{R}^i_{\overline{\pi}}$ with unlimited number of successive times attaining positive reward, is higher than the expected return of any other trace. With unlimited number of obtaining positive reward for the traces entering the accepting recurrent class $\mathfrak{R}^i_{\overline{\pi}}$, and with a state-dependent discount factor, it can be shown that the expected return is bounded and is higher than that for non-accepting traces, which have limited number of attainment of positive rewards.

In the following, by contradiction, we show that any optimal policy π^* which optimises the expected return will satisfy the property with maximum probability if η is close to one. Recall from Definition 9 that the probability of satisfying φ under policy π at state $s^{\otimes}$ is:

$$Pr(s^{\otimes}..^{\pi} \models \varphi),$$

where $s^{\otimes}..^{\pi}$ is the collection of all paths starting from $s^{\otimes}$ under policy π. Thus, for the policy $\overline{\pi}$ we have:

$$\overline{\pi} = \arg\sup_{\pi \in \mathcal{D}} Pr(s^{\otimes}..^{\pi} \models \varphi). \tag{13}$$

Accordingly, the expected return for for policy $\overline{\pi}$ can be rewritten as:

$$\begin{aligned} &U^{\overline{\pi}}(s^{\otimes}) = \\ &\mathbb{E}^{\overline{\pi}}\Big[\sum_{n=0}^{\infty} \gamma(s_n^{\otimes})^{N(s_n^{\otimes})}\ r_n \Big| s_0^{\otimes} = s^{\otimes},\ \rho = L(s_0)L(s_1)... \models \varphi\Big] Pr(s^{\otimes}..^{\overline{\pi}} \models \varphi) + \\ &\mathbb{E}^{\overline{\pi}}\Big[\sum_{n=0}^{\infty} \gamma(s_n^{\otimes})^{N(s_n^{\otimes})}\ r_n \Big| s_0^{\otimes} = s^{\otimes},\ \rho = L(s_0)L(s_1)... \not\models \varphi\Big] Pr(s^{\otimes}..^{\overline{\pi}} \not\models \varphi). \end{aligned} \tag{14}$$

Of course, when the policy traces satisfy the property, with unlimited number of positive reward attainment the expected return under (2) is

$$\begin{aligned} &\mathbb{E}^{\overline{\pi}}\Big[\sum_{n=0}^{\infty} \gamma(s_n^{\otimes})^{N(s_n^{\otimes})}\ r_n \Big| s_0^{\otimes} = s^{\otimes},\ \rho = L(s_0)L(s_1)... \models \varphi\Big] \times \\ &Pr(s^{\otimes}..^{\overline{\pi}} \models \varphi) = \frac{r_p}{1-\eta} Pr(s^{\otimes}..^{\overline{\pi}} \models \varphi). \end{aligned} \tag{15}$$

Once the induced traces do not satisfy the property we have:

$$\begin{aligned} &\mathbb{E}^{\overline{\pi}}\Big[\sum_{n=0}^{\infty} \gamma(s_n^{\otimes})^{N(s_n^{\otimes})}\ r_n \Big| s_0^{\otimes} = s^{\otimes},\ \rho = L(s_0)L(s_1)... \not\models \varphi\Big] \times \\ &Pr(s^{\otimes}..^{\overline{\pi}} \not\models \varphi) = \sum_{\rho \in L(s^{\otimes}..^{\overline{\pi}} \not\models \varphi)} Pr(\rho \in L(s^{\otimes}..^{\overline{\pi}} \not\models \varphi)) \frac{r_p(1-\eta^{|\overline{J}|_\rho})}{1-\eta}, \end{aligned} \tag{16}$$

where $|\overline{J}|_\rho$ is the (finite) number of times that the trace ρ intersected with the accepting frontier set $\mathbb{A}$ and the agent received a positive reward (see (12)).

From (15) and (16) we can reformulate (14) as follows:

$$U^{\overline{\pi}}(s^{\otimes}) = \frac{r_p}{1-\eta} Pr(s^{\otimes}..^{\overline{\pi}} \models \varphi) + \sum_{\rho \in L(s^{\otimes}..^{\overline{\pi}} \not\models \varphi)} Pr(\rho \in L(s^{\otimes}..^{\overline{\pi}} \not\models \varphi)) \frac{r_p(1-\eta^{|\overline{J}|_\rho})}{1-\eta}. \tag{17}$$

Similarly for the optimal policy π^* we have

$$U^{\pi^*}(s^{\otimes}) = \frac{r_p}{1-\eta} Pr(s^{\otimes}..^{\pi^*} \models \varphi) + \sum_{\rho \in L(s^{\otimes}..^{\pi^*} \not\models \varphi)} Pr(\rho \in L(s^{\otimes}..^{\pi^*} \not\models \varphi)) \frac{r_p(1-\eta^{|J^*|_\rho})}{1-\eta}, \tag{18}$$

where $|J^*|_\rho$ is the (finite) number of times that the trace ρ intersected with the accepting frontier set $\mathbb{A}$. We then factorise $r_p/1-\eta$ from (17) and (18):

$$U^{\overline{\pi}}(s^{\otimes}) = \frac{r_p}{1-\eta}\Big[Pr(s^{\otimes}..^{\overline{\pi}} \models \varphi) + \sum_{\rho \in L(s^{\otimes}..^{\overline{\pi}} \not\models \varphi)} Pr(\rho \in L(s^{\otimes}..^{\overline{\pi}} \not\models \varphi))(1-\eta^{|\overline{J}|_\rho})\Big]. \tag{19}$$

$$U^{\pi^*}(s^{\otimes}) = \frac{r_p}{1-\eta}\Big[Pr(s^{\otimes}..^{\pi^*} \models \varphi) + \sum_{\rho \in L(s^{\otimes}..^{\pi^*} \not\models \varphi)} Pr(\rho \in L(s^{\otimes}..^{\pi^*} \not\models \varphi))(1-\eta^{|J^*|_\rho})\Big]. \tag{20}$$

Now suppose that the optimal policy π^* does not satisfy the property φ with maximum probability. Given that $\overline{\pi}$ maximises the satisfaction probability, from (13) we have:

$$Pr(s^{\otimes}..^{\overline{\pi}} \models \varphi) > Pr(s^{\otimes}..^{\pi^*} \models \varphi). \tag{21}$$

At the same time, it is easy to see from (19) and (20) that:

$$\lim_{\eta \to 1^-} \frac{U^{\overline{\pi}}(s^{\otimes})}{U^{\pi^*}(s^{\otimes})} = \frac{Pr(s^{\otimes}..^{\overline{\pi}} \models \varphi)}{Pr(s^{\otimes}..^{\pi^*} \models \varphi)},$$

and consequently from (21):

$$U^{\overline{\pi}}(s^{\otimes}) > U^{\pi^*}(s^{\otimes})$$

This is, however, in direct contrast with Definition 4 and the optimality of the policy π^*, leading to a contradiction. This essentially means that the optimal policy π^* maximises the probability of satisfying φ.

Corollary 2 (Maximum Probability of Satisfaction)**.** *From Definition 9, for a discounting factor close enough to* 1 *the maximum probability of satisfaction at any state* $s^{\otimes}$ *can be determined from the LCRL value function as*

$$Pr_{\max}(s^{\otimes} \models \varphi) = \frac{1-\eta}{r_p}\ U^{\pi^*}(s^{\otimes}).$$

proof.
The proof is a direct consequent of (20) and Theorem 3 results when $\eta \to 1^{-}$.

Corollary 3. *If no policy in the MDP $\mathfrak{M}$ can be generated to satisfy the property φ, LCRL yields in the limit a policy that is closest (according to the previous Definition) to satisfying the given LTL formula φ.*

proof.
Assume that there exists no policy in the MDP $\mathfrak{M}$ that can satisfy the property φ. Construct the induced Markov chain $\mathfrak{M}_{\mathfrak{A}}^{\pi}$ for any arbitrary policy π and its associated set of transient states $\mathfrak{T}_{\pi}$ and h sets of irreducible recurrent classes $\mathfrak{R}_{\pi}^{i}$:

$$\mathfrak{M}_{\mathfrak{A}}^{\pi} = \mathfrak{T}_{\pi} \sqcup \mathfrak{R}_{\pi}^{1} \sqcup ... \sqcup \mathfrak{R}_{\pi}^{h}.$$

By assumption, policy π cannot satisfy the property and we thus have that

$$\forall \mathfrak{R}_{\pi}^{i},\ \exists j \in \{1, ..., f\},\ F_{j}^{\otimes} \cap \mathfrak{R}_{\pi}^{i} = \emptyset,$$

which means that there are some automaton accepting sets like F_j that cannot be visited. Therefore, after a limited number of times no positive reward is given by the reward function $R(s^{\otimes}, a)$. However, the closest recurrent class to satisfying the property is the one that intersects with more distinct accepting sets. More specifically, this is the recurrent class whose runs can partially satisfy φ. From the LDGBA construction [126], by examining the accepting sets that are visited in this recurrent class we can also determine which sub-formula of φ is satisfiable by π.

By Definition 3, for any arbitrary $r_p > 0$ (and $r_n = 0$), the expected return at the initial state for a trace with highest number of intersections with distinct accepting sets is maximum among other traces. Hence, an optimal policy produced by LCRL converges to a policy whose recurrent classes of its induced Markov chain have the highest number of intersections with the accepting sets of the automaton.

B. Partial Satisfaction

Definition 16 (Closeness to Satisfaction). Assume that the probability of satisfying the property φ for two policies π_1 and π_2 is zero. Accordingly, there are accepting sets in the automaton that have no intersection with runs of induced Markov chains $\mathfrak{M}^{\pi_1}$ and $\mathfrak{M}^{\pi_2}$. We say that π_1 is closer to satisfying the property if runs of $\mathfrak{M}^{\pi_1}$ cross a larger number of distinct accepting sets of the automaton, than runs of $\mathfrak{M}^{\pi_2}$. ⌟

In the following we show that even when it is infeasible to satisfy the given LTL specification, LCRL is able to find a policy whose traces are closest to satisfying φ. Specifically, depending on whether some accepting sets can be visited infinitely often, the LTL property can be partially satisfied, i.e. a sub-formula of φ.

Corollary 4. *If no policy in the MDP $\mathfrak{M}$ can be generated to satisfy the property φ, LCRL yields in the limit a policy that is closest (according to the previous Definition) to satisfying the given LTL formula φ.*

proof.
Assume that there exists no policy in the MDP $\mathfrak{M}$ that can satisfy the property φ. Construct the induced Markov chain $\mathfrak{M}_{\mathfrak{A}}^{\pi}$ for any arbitrary policy π and its associated set of transient states $\mathfrak{T}_{\pi}$ and h sets of irreducible recurrent classes $\mathfrak{R}_{\pi}^{i}$:

$$\mathfrak{M}_{\mathfrak{A}}^{\pi} = \mathfrak{T}_{\pi} \sqcup \mathfrak{R}_{\pi}^{1} \sqcup ... \sqcup \mathfrak{R}_{\pi}^{h}.$$

By assumption, policy π cannot satisfy the property and we thus have that

$$\forall \mathfrak{R}_{\pi}^{i},\ \exists j \in \{1, ..., f\},\ F_j^{\otimes} \cap \mathfrak{R}_{\pi}^{i} = \emptyset,$$

which means that there are some automaton accepting sets like F_j that cannot be visited. Therefore, after a limited number of times no positive reward is given by the reward function $R(s^{\otimes}, a)$. However, the closest recurrent class to satisfying the property is the one that intersects with more distinct accepting sets. More specifically, this is the recurrent class whose runs can partially satisfy φ. From the LDGBA construction [126], by examining the accepting sets that are visited in this recurrent class we can also determine which sub-formula of φ is satisfiable by π.

By Definition 3, for any arbitrary $r_p > 0$ (and $r_n = 0$), the expected return at the initial state for a trace with highest number of intersections with distinct accepting sets is maximum among other traces. Hence, an optimal policy produced by LCRL converges to a policy whose recurrent classes of its induced Markov chain have the highest number of intersections with the accepting sets of the automaton.

C. Comparison with a DRA-based Learning Algorithm

The problem of policy synthesis with RL under LTL constraints is investigated in several previous works, where the general recipe is to translate the LTL property into a DRA and then to construct a product MDP. For the sake of comparison, let us consider the example in [119]: a 5×5 grid world where the starting state is $(0, 3)$and the agent has to visit two regions infinitely often (areas A and B in Fig. 10). The agent has to also avoid the area C. This property can be encoded as the following LTL formula:

$$\Box\Diamond A \wedge \Box\Diamond B \wedge \Box\neg C. \tag{22}$$

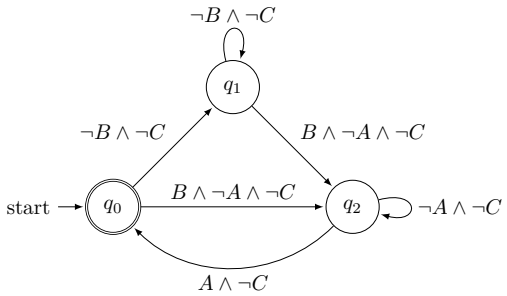


Figure 9: LDGBA expressing the LTL formula in (22) with removed transitions labelled $A \wedge B$ (since it is impossible to be at A and B at the same time).

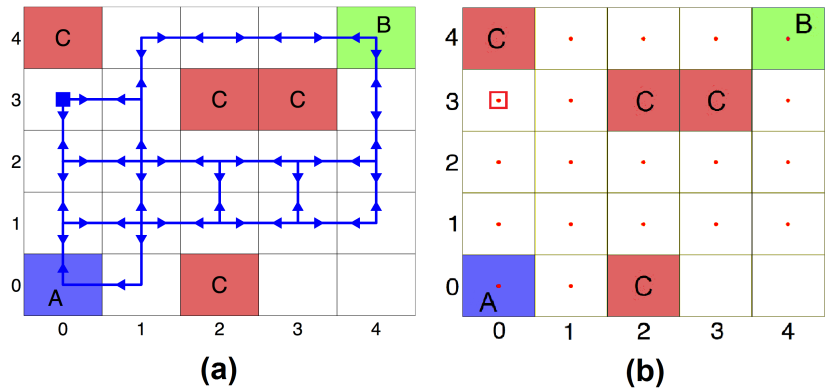


Figure 10: (a) Example considered in [119]. (b) Trajectories under the policy generated by LCRL in [119].

The product MDP in [119] contains 150 states, which means that the Rabin automaton has 6 states. Fig. 10.a shows the trajectories under the optimal policy generated by [119] algorithm after 600 iterations. However, by employing LCRL

we are able to generate the same trajectories with only 50 iterations (Fig. 10.b). The automaton that we consider is an LDGBA with only 3 states as in Fig. 9. This result in a smaller product MDP and a much more succinct state space (only 75 states) for the algorithm to learn, which consequently leads to a faster convergence.

In addition, the reward shaping in LCRL is significantly simpler thanks to the Büchi acceptance condition. In a DRA $\mathbf{R}(\mathcal{Q}, \mathcal{Q}_0, \Sigma, \mathcal{F}, \Delta)$, the set $\mathcal{F} = \{(G_1, B_1), \ldots, (G_{n_F}, B_{n_F})\}$ represents the acceptance condition in which G_i, $B_i \in \mathcal{Q}$ for $i = 1, \ldots, n_F$. An infinite run $\theta \in \mathcal{Q}^\omega$ starting from $\mathcal{Q}_0$ is accepting if there exists $i \in \{1, \ldots, n_F\}$ such that

$$inf(\theta) \cap G_i \neq \emptyset \quad \text{and} \quad inf(\theta) \cap B_i = \emptyset.$$

Therefore for each $i \in \{1, \ldots, n_F\}$ a separate reward assignment is needed in [119] which complicates the implementation and increases the required calculation costs. This complicated reward assignment is not needed by employing the accepting frontier function in our scheme.

More importantly, LCRL is a model-free learning algorithm that does not require an approximation of the transition probabilities of the underlying MDP. This even makes LCRL more easier to employ. We would like to emphasize that LCRL convergence proof solely depends on the structure of the MDP and this allows LCRL to find satisfying policies even if they have probability of less than one.

D. Alternatives to LCNFQ

In the following, we discuss the most popular alternative approaches to solving infinite-state MDPs, namely the Voronoi Quantiser (VQ) and Fitted Value Iteration (FVI).

D.1. Voronoi Quantiser

VQ can be classified as a discretisation algorithm which abstracts the continuous-state MDP to a finite-state MDP, allowing classical RL to be run. However, most of discretisation techniques are usually done in an ad-hoc manner, disregarding one of the most appealing features of RL: autonomy. In other words, RL is able to produce the optimal policy with regards to the reward function, with minimum supervision. Therefore, the state space discretisation should be performed as part of the learning task, instead of being fixed at the start of the learning process.

Inspired by [90], we propose a version of VQ that is able to discretise the state space of the product MDP $\mathcal{S}^{\otimes}$, while allowing RL to explore the MDP.

Algorithm 4: Episodic VQ

input : minimum resolution Δ
output : approximated Q-function Q

initialize $c_1 = c =$ initial state
initialize $Q(c_1, a) = 0,\ \forall a \in \mathcal{A}$
repeat
  set $\mathcal{C} = c_1$
  $a_* = \arg\max_{a \in \mathcal{A}} Q(c, a)$
  repeat
    execute action a_* and observe the next state (s', q)
    if $\mathcal{C}^q$ *is empty* **then**
      append $c_{new} = (s', q)$ to $\mathcal{C}^q$
      initialize $Q(c_{new}, a) = 0,\ \forall a \in \mathcal{A}$
    else
      determine the nearest neighbour c_{new} within $\mathcal{C}^q$
      if $c_{new} = c$ **then**
        if $||c - (s', q)||_2 > \Delta$ **then**
          append $c_{new} = (s', q)$ to $\mathcal{C}^q$
          initialize $Q(c_{new}, a) = 0,\ \forall a \in \mathcal{A}$
        end
      else
        $Q(c, a_*) = (1 - \mu)Q(c, a_*) + \mu[R(c, a_*) + \gamma \max_{a'}(Q(c_{new}, a'))]$
      end
    end
    $c = c_{new}$
  until *end of trial*
until *end of trial*

VQ maps the state space onto a finite set of disjoint regions called Voronoi cells [145]. The set of centroids of these cells is denoted by $\mathcal{C} = \{c_i\}_{i=1}^{m},\ c_i \in \mathcal{S}^{\otimes}$, where m is the number of the cells. With $\mathcal{C}$, we are able to use QL and find an approximation of the optimal policy for a continuous-state MDP.

In the beginning, $\mathcal{C}$ is initialised to consist of just one c_1, which corresponds to the initial state. This means that the agent views the entire state space as a homogeneous region when no a-priori knowledge is available. Assuming that states are represented by vectors, when the agent explores this unknown state space, the Euclidean norm of the distance between each newly visited state and its nearest neighbour can calculated. If this norm is greater than a threshold value Δ called "minimum resolution", or if the new state $s^{\otimes}$ comprises an automaton state that has never been visited, then the newly visited state is appended to $\mathcal{C}$. Therefore, as the agent continues to explore, the size of $\mathcal{C}$ would increase until the "relevant" parts of the state space are partitioned. In our algorithm, the set $\mathcal{C}$ has $|\mathcal{Q}|$ disjoint subsets where $\mathcal{Q}$ is the finite set of states of the automaton. Each subset $\mathcal{C}^{q_j},\ j = 1, ..., |\mathcal{Q}|$ contains the centroids of those Voronoi cells that have the form of $c_i^{q_j} = (\cdot, q_j)$, i.e. $\bigcup_i^m c_i^{q_j} = \mathcal{C}^{q_j}$ and $\mathcal{C} = \bigcup_{j=1}^{|\mathcal{Q}|} \mathcal{C}^{q_j}$. Therefore, a Voronoi cell

$$\{(s, q_j) \in \mathcal{S}^{\otimes}, ||(s, q_j) - c_i^{q_j}||_2 \leq ||(s, q_j) - c_{i'}^{q_j}||_2\},$$

is defined by the nearest neighbour rule for any $i' \neq i$. The proposed VQ algorithm is presented in Algorithm 4.

D.2. Fitted Value Iteration

FVI is an approximate DP algorithm for continuous-state MDPs, which employs function approximation techniques [49]. In standard DP the goal is to find a mapping, i.e. value function, from the state space to $\mathbb{R}$, which can generate the optimal policy. The value function in our setup is the expected reward in (1) when π is the optimal policy, i.e. U^{π^*}. Over continuous state spaces, analytical representations of the value function are in general not available. Approximations can be obtained numerically through approximate value iterations, which involve approximately iterating a Bellman operator on a an approximate value function [131].

We propose a modified version of FVI that can handle the product MDP. The global value function $v : \mathcal{S}^{\otimes} \to \mathbb{R}$, or more specifically $v : \mathcal{S} \times \mathcal{Q} \to \mathbb{R}$, consists of $|\mathcal{Q}|$ number of components. For each $q_j \in \mathcal{Q}$, the sub-value function $v^{q_j} : \mathcal{S} \to \mathbb{R}$ returns the value the states of the form (s, q_j). Similar to the LCNFQ algorithm, the components are not decoupled.

Let $P^{\otimes}(dy|s^{\otimes}, a)$ be the distribution over $\mathcal{S}^{\otimes}$ for the successive state given that the current state is $s^{\otimes}$ and the selected action is a. For each state (s, q_j), the Bellman update over each component of value function v^{q_j} is defined as:

$$\tau v^{q_j}(s) = \sup_{a \in \mathcal{A}} \{ \int v(y) P^{\otimes}(dy|(s, q_j), a) \}, \tag{23}$$

Algorithm 5: FVI

input : MDP $\mathfrak{M}$, a set of samples $\{s_i^{\otimes}\}_{i=1}^k = \{(s_i, q_j)\}_{i=1}^k$ for each $q_j \in \mathcal{Q}$, Monte Carlo sampling number Z, smoothing parameter h'

output : approximated value function Lv

initialize Lv

sample $\mathcal{Y}_a^Z(s_i, q_j),\ \forall q_j \in \mathcal{Q},\ \forall i = 1, ..., k\ ,\ \forall a \in \mathcal{A}$

repeat

  for $j = |\mathcal{Q}|$ ***to*** 1 **do**

    $\forall q_j \in \mathcal{Q},\ \forall i = 1, ..., k\ ,\ \forall a \in \mathcal{A}$ calculate $I_a((s_i, q_j)) = 1/Z \sum_{y \in \mathcal{Y}_a^Z(s_i, q_j)} Lv(y)$ using (25)

    for each state (s_i, q_j), update $v^{q_j}(s_i) = \sup_{a \in \mathcal{A}}\{I_a((s_i, q_j))\}$ in (25)

  end

until *end of trial*

where τ is the Bellman operator [69]. The update in (23) is different than the standard Bellman update in DP, as it does not comprise a running reward, and as the (terminal) reward is replaced by the following function initialization:

$$v(s^{\otimes}) = \begin{cases} r_p & \text{if } s^{\otimes} \in \mathbb{A}, \\ r_n & \text{otherwise.} \end{cases} \tag{24}$$

The main hurdle in executing the Bellman operator in continuous state MDPs, as in (23), is that no analytical representations of the value function v and of its components $v^{q_j},\ q_j \in \mathcal{Q}$ are in general available. Therefore, we employ an approximation method, by introducing a new operator L.

The operator L provides an approximation of the value function, denoted by Lv, and of its components v^{q_j}, which we denote by Lv^{q_j}. For each $q_j \in \mathcal{Q}$ the approximation is based on a set of points $\{(s_i, q_j)\}_{i=1}^k \subset \mathcal{S}^{\otimes}$ which are called centres. For each q_j, the centres $i = 1, ..., k$ are distributed uniformly over $\mathcal{S}$.

In the proposed FVI algorithm, we employ the kernel averager method [131], which can be represented by the following expression for each state (s, q_j):

$$Lv(s, q_j) = Lv^{q_j}(s) = \frac{\sum_{i=1}^k K(s_i - s) v^{q_j}(s_i)}{\sum_{i=1}^k K(s_i - s)}, \tag{25}$$

where the kernel $K : \mathcal{S} \to \mathbb{R}$ is a radial basis function, such as $e^{-|s - s_i|/h'}$, and h' is smoothing parameter. Each kernel is characterised by the point s_i, and its value decays to zero as s diverges from s_i. This means that for each $q_j \in \mathcal{Q}$ the approximation operator L in (25) is a convex combination of the values of the centres $\{s_i\}_{i=1}^k$ with larger weight given to those values $v^{q_j}(s_i)$ for which s_i is close to s. Note that the smoothing parameter h' controls the weight assigned to more distant values.

In order to approximate the integral in the Bellman update (23) we use a Monte Carlo sampling technique [125]. For each centre (s_i, q_j) and for each action a, we sample the next state $y_a^z(s_i, q_j)$ for $z = 1, ..., Z$ times and append

these samples to the set of Z subsequent states $\mathcal{Y}_a^Z(s_i, q_j)$. We then replace the integral with

$$I_a(s_i, q_j) = \frac{1}{Z} \sum_{z=1}^{Z} Lv(y_a^z(s_i, q_j)). \tag{26}$$

The approximate value function Lv is initialised according to (24). In each loop in FVI, the Bellman update approximation is first executed over those sub-value functions that are linked with the accepting states of the LDGBA, i.e. those that have an initial value of r_p. The approximate Bellman update then goes backward until it reaches those sub-value functions that are linked with the initial states of the automaton. This allows the state values to back-propagate through the product MDP transitions that connects the sub-value function via (26). Without loss of generality we assume that the automaton states are ordered and hence the back-propagation starts from $q_i = |\mathcal{Q}|$. Once we have the approximated value function, we can generate the optimal policy by following the maximum value (Algorithm 5).

We conclude this section by emphasising that Algorithms 4 and 5 are proposed to be benchmarked against LCNFQ. Further, MDP abstraction techniques such as [130] failed to scale and to find an optimal policy.

References

[1] A. Abate, M. Prandini, J. Lygeros, and S. Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. *NeurIPS*, 19:1, 2007.

[3] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. *arXiv preprint:1708.08611*, 2017.

[4] R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *TOCL*, 5(1):1–25, 2004.

[5] O. Andersson, F. Heintz, and P. Doherty. Model-based reinforcement learning in continuous environments using real-time constrained optimization. In *AAAI Conference on Artificial Intelligence*, pages 2497–2503, 2015.

[6] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175, 2017.

[7] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, volume 70, pages 166–175, 2017.

[8] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *AAAI Conference on Artificial Intelligence*, 2017.

[9] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[10] S. Bansal, L. Kavraki, M. Y. Vardi, and A. Wells. Synthesis from satisficing and temporal goals. *arXiv preprint arXiv:2205.10464*, 2022.

[11] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[12] L. Belzner and M. Wirsing. Synthesizing safe policies under probabilistic constraints with reinforcement learning and Bayesian model checking. *arXiv preprint:2005.03898*, 2020.

[13] D. Bertsekas. Convergence of discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, 20(3):415–419, 1975.

[14] D. P. Bertsekas and S. Shreve. *Stochastic optimal control: the discrete-time case*. Athena Scientific, 2004.

[15] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic Programming*, volume 1. Athena Scientific, 1996.

[16] S. Bharadwaj, S. Le Roux, G. Pérez, and U. Topcu. Reduction techniques for model checking and learning in MDPs. In *International Joint Conference on Artificial Intelligence*, pages 4273–4279, 2017.

[17] F. Blahoudek, M. Heizmann, S. Schewe, J. Strejček, and M.-H. Tsai. Complementing semi-deterministic Büchi automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 770–787. Springer, 2016.

[18] J. Bogdoll, L. M. F. Fioriti, A. Hartmanns, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *Formal Techniques for Distributed Systems*, pages 59–74. Springer, 2011.

[19] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. *arXiv preprint:1909.07299*, 2019.

[20] T. Brázdil, K. Chatterjee, M. Chmelík, V. Forejt, J. Křetínskỳ, M. Kwiatkowska, D. Parker, and M. Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, pages 98–114. Springer, 2014.

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. *arXiv preprint:1606.01540*, 2016.

[22] M. Cai, M. Hasanbeig, S. Xiao, A. Abate, and Z. Kan. Modular deep reinforcement learning for continuous motion planning with temporal logic. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) and IEEE Robotics and Automation letters*, 2021.

[23] M. Cai, Z. Li, H. Gao, S. Xiao, and Z. Kan. Optimal probabilistic motion planning with partially infeasible LTL constraints. *arXiv preprint:2007.14325*, 2020.

[24] M. Cai, H. Peng, Z. Li, H. Gao, and Z. Kan. Receding horizon control based online motion planning with partially infeasible LTL specifications. *arXiv preprint:2007.12123*, 2020.

[25] M. Cai, H. Peng, Z. Li, and Z. Kan. Learning-based probabilistic LTL motion planning with environment and motion uncertainties. *IEEE Trans. Autom. Control*, 66(5):2386–2392, 2021.

[26] M. Cai and C.-I. Vasile. Safety-critical learning of robot control with temporal logic specifications. *arXiv preprint arXiv:2109.02791*, 2021.

[27] M. Cai, S. Xiao, and Z. Kan. Reinforcement learning based temporal logic control with soft constraints using limit-deterministic Büchi automata. *arXiv preprint:2101.10284*, 2021.

[28] A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith. Non-Markovian rewards expressed in LTL: guiding search via reward shaping. In *Tenth Annual Symposium on Combinatorial Search*, 2017.

[29] A. Camacho, R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 6065–6073, 2019.

[30] S. Carr, N. Jansen, S. Junges, and U. Topcu. Safe reinforcement learning via shielding for POMDPs. *arXiv preprint arXiv:2204.00755*, 2022.

[31] N. Cauchi and A. Abate. Stochy-automated verification and synthesis of stochastic processes. In *HSCC*, pages 258–259, 2019.

[32] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith. *Model checking*. MIT Press, 2018.

[33] S. P. Coraluppi and S. I. Marcus. Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes. *Automatica*, 35(2):301–309, 1999.

[34] C. Daniel, G. Neumann, and J. Peters. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pages 273–281, 2012.

[35] A. David, P. G. Jensen, K. G. Larsen, M. Mikučionis, and J. H. Taankvist. Uppaal Stratego. In *TACAS*, pages 206–211. Springer, 2015.

[36] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and Z. Wang. Time for statistical model checking of real-time systems. In *Computer Aided Verification*, pages 349–355. Springer, 2011.

[37] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[38] F. Dufour and T. Prieto-Rumeau. Approximation of Markov decision processes with general state space. *Journal of Mathematical Analysis and Applications*, 388(2):1254–1267, 2012.

[39] R. Durrett. *Essentials of stochastic processes*, volume 1. Springer, 1999.

[40] J. Fu and U. Topcu. Probably approximately correct MDP learning and control with temporal logic constraints. In *Robotics: Science and Systems X*, 2014.

[41] N. Fulton. *Verifiably Safe Autonomy for Cyber-Physical Systems*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2018.

[42] N. Fulton, N. Hunt, N. Hoang, and S. Das. Formal verification of end-to-end learning in cyber-physical systems: Progress and challenges. *arXiv preprint:2006.09181*, 2020.

[43] N. Fulton and A. Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *AAAI Conference on Artificial Intelligence*, pages 6485–6492, 2018.

[44] N. Fulton and A. Platzer. Verifiably safe off-model reinforcement learning. In *TACAS*, pages 413–430, 2019.

[45] J. Garcia and F. Fernández. Safe exploration of state and action spaces in reinforcement learning. *Journal of Artificial Intelligence Research*, 45:515–564, 2012.

[46] J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. *JMLR*, 16(1):1437–1480, 2015.

[47] P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108, 2005.

[48] M. Giacobbe, M. Hasanbeig, D. Kroening, and H. Wijk. Shielding Atari games with bounded prescience. In *Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1507–1509. ACM, 2021.

[49] G. J. Gordon. Stable function approximation in dynamic programming. In *Machine Learning*, pages 261–268. Elsevier, 1995.

[50] D. Grbic and S. Risi. Safe reinforcement learning through meta-learned instincts. In *Artificial Life Conference Proceedings*, pages 283–291. MIT Press, 2020.

[51] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[52] E. Gunter. From natural language to linear temporal logic: Aspects of specifying embedded systems in LTL. In *Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation*, 2003.

[53] S. Haesaert, S. E. Zadeh Soudjani, and A. Abate. Verification of general Markov decision processes by approximate similarity relations and policy refinement. *SIAM Journal on Control and Optimization*, 55(4):2333–2367, 2017.

[54] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *TACAS*, pages 395–412. Springer, 2019.

[55] L. Hammond, A. Abate, J. Gutierrez, and M. J. Wooldridge. Multi-agent reinforcement learning with temporal logic specifications. In *Autonomous Agents and Multiagent Systems*, pages 583–592. ACM, 2021.

[56] D. Harris and S. Harris. *Digital Design and Computer Architecture*. Morgan Kaufmann, 2010.

[57] H. Hasanbeig. *Safe and certified reinforcement learning with logical constraints*. PhD thesis, University of Oxford, 2020.

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

[59] M. Hasanbeig, A. Abate, and D. Kroening. Certified reinforcement learning with logic guidance. *arXiv preprint:1902.00778*, 2019.

[60] M. Hasanbeig, A. Abate, and D. Kroening. Logically-constrained neural fitted Q-iteration. In *AAMAS*, pages 2012–2014. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[61] M. Hasanbeig, A. Abate, and D. Kroening. Cautious reinforcement learning with logical constraints. In *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[62] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *Conference on Decision and Control*, pages 5338–5343. IEEE, 2019.

[63] M. Hasanbeig, D. Kroening, and A. Abate. Deep reinforcement learning with temporal logics. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–22. Springer, 2020.

[64] M. Hasanbeig, D. Kroening, and A. Abate. Towards verifiable and safe model-free reinforcement learning. In *Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY)*, pages 1–10. Italian Association for Artificial Intelligence, 2020.

[65] M. Hasanbeig, D. Kroening, and A. Abate. LCRL: Certified policy synthesis via logically-constrained reinforcement learning. In *Quantitative Evaluation of Systems (QEST)*, pages 217–231. Springer, 2022.

[66] M. Hasanbeig, N. Yogananda Jeppu, A. Abate, T. Melham, and D. Kroening. DeepSynth: Automata synthesis for automatic task segmentation in deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, pages 7647–7656. Association for the Advancement of Artificial Intelligence, 2021.

[67] M. Hausknecht and P. Stone. Deep recurrent Q-learning for partially observable MDPs. In *2015 AAAI Fall Symposium Series*, pages 29–37, 2015.

[68] D. Henriques, J. G. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. Statistical model checking for Markov decision processes. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*, pages 84–93. IEEE, 2012.

[69] O. Hernández-Lerma and J. B. Lasserre. *Further topics on discrete-time Markov control processes*, volume 42. Springer Science & Business Media, 2012.

[70] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[71] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[72] T. Jaakkola, M. I. Jordan, and S. P. Singh. Convergence of stochastic iterative dynamic programming algorithms. In *NeurIPS*, pages 703–710, 1994.

[73] N. Jansen, B. Könighofer, S. Junges, and R. Bloem. Shielded decision-making in MDPs. *arXiv preprint:1807.06096*, 2018.

[74] S. Junges, N. Jansen, C. Dehnert, U. Topcu, and J.-P. Katoen. Safety-constrained reinforcement learning for MDPs. In *TACAS*, pages 130–146. Springer, 2016.

[75] S. Kalluraya, G. J. Pappas, and Y. Kantaros. Resilient temporal logic planning in the presence of robot failures. *arXiv preprint arXiv:2305.05485*, 2023.

[76] Y. Kantaros. Accelerated reinforcement learning for temporal logic control objectives. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5077–5082. IEEE, 2022.

[77] P. Kapoor, A. Balakrishnan, and J. V. Deshmukh. Model-based reinforcement learning from signal temporal logic specifications. *arXiv preprint:2011.04950*, 2020.

[78] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

[79] D. Kini and M. Viswanathan. Limit deterministic and probabilistic automata for LTL\GU. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 628–642. Springer, 2015.

[80] D. Kini and M. Viswanathan. Optimal translation of LTL to limit deterministic automata. In *TACAS*, pages 113–129. Springer, 2017.

[81] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS*, pages 3675–3683, 2016.

[82] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[83] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification*, pages 585–591. Springer, 2011.

[84] M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *ICRA*, pages 3227–3232. IEEE, 2010.

[85] K. G. Larsen. Priced timed automata and statistical model checking. In *International Conference on Integrated Formal Methods*, pages 154–161. Springer, 2013.

[86] R. Lassaigne and S. Peyronnet. Approximate planning and verification for large Markov decision processes. *International Journal on Software Tools for Technology Transfer*, 17(4):457–467, 2015.

[87] A. Lavaei, M. Khaled, S. Soudjani, and M. Zamani. AMYTISS: Parallelized automated controller synthesis for large-scale stochastic systems. In *Computer Aided Verification*, pages 461–474. Springer, 2020.

[88] A. Lavaei, F. Somenzi, S. Soudjani, A. Trivedi, and M. Zamani. Formal controller synthesis for continuous-space MDPs via model-free reinforcement learning. In *ICCPS*, pages 98–107. IEEE, 2020.

[89] A. Lavaei, S. Soudjani, A. Abate, and M. Zamani. Automated verification and synthesis of stochastic hybrid systems: A survey. *Automatica, arXiv preprint:2101.07491*, 2022.

[90] I. S. Lee and H. Y. Lau. Adaptive state space partitioning for reinforcement learning. *Engineering Applications of Artificial Intelligence*, 17(6):577–588, 2004.

[91] A. Legay, S. Sedwards, and L.-M. Traonouez. Scalable verification of Markov decision processes. In *International Conference on Software Engineering and Formal Methods*, pages 350–362. Springer, 2014.

[92] B. Lennartson and Q. Jia. Reinforcement learning with temporal logic constraints. In *International Workshop on Discrete Event Systems (WODES)*, 2020.

[93] K. Lesser and A. Abate. Multiobjective optimal control with safety as a priority. *IEEE Transactions on Control Systems Technology*, 26(3):1015–1027, 2018.

[94] X. Li, C.-I. Vasile, and C. Belta. Reinforcement learning with temporal logic rewards. *arXiv preprint:1612.03471*, 2016.

[95] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

[96] J. Lope and J. Martin. Learning autonomous helicopter flight with evolutionary reinforcement learning. In *International Conference on Computer Aided Systems Theory*, pages 75–82. Springer, 2009.

[97] F. Memarian, Z. Xu, B. Wu, M. Wen, and U. Topcu. Active task-inference-guided deep inverse reinforcement learning. In *CDC*, pages 1932–1938. IEEE, 2020.

[98] C. Mingyu, S. Xiao, Z. Li, and Z. Kan. Optimal probabilistic motion planning with potential infeasible LTL constraints. *IEEE Transactions on Automatic Control*, 2021.

[99] R. N. L. Mitta, H. Hasanbeig, D. Kroening, and A. Abate. Risk-aware Bayesian reinforcement learning for cautious exploration. In *NeurIPS ML Safety Workshop*, 2022.

[100] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint:1312.5602*, 2013.

[101] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[102] R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49(2-3):291–323, 2002.

[103] R. G. Newell and W. A. Pizer. Discounting the distant future: how much do uncertain rates increase valuations? *Journal of Environmental Economics and Management*, 46(1):52–71, 2003.

[104] A. P. Nikora and G. Balcom. Automated identification of LTL patterns in natural language requirements. In *ISSRE*, pages 185–194. IEEE, 2009.

[105] S. Pathak, L. Pulina, and A. Tacchella. Verification and repair of control policies for safe reinforcement learning. *Applied Intelligence*, pages 1–23, 2017.

[106] M. Pecka and T. Svoboda. Safe exploration techniques for reinforcement learning–an overview. In *International Workshop on Modelling and Simulation for Autonomous Systems*, pages 357–375. Springer, 2014.

[107] J. Peters and J. A. Bagnell. Policy gradient methods. *Scholarpedia*, 5(11):3698, 2010.

[108] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264. IEEE, 2006.

[109] S. Pitis. Rethinking the discount factor in reinforcement learning: A decision theoretic approach. *arXiv preprint:1902.02893*, 2019.

[110] A. Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 41(2):143–189, 2008.

[111] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[112] K. Polymenakos, A. Abate, and S. Roberts. Safe policy search with Gaussian process models. *arXiv preprint:1712.05556*, 2017.

[113] M. Prandini and J. Hu. A stochastic approximation method for reachability computations. In *Stochastic Hybrid Systems*, pages 107–139. Springer, 2006.

[114] D. Precup. *Temporal abstraction in reinforcement learning.* PhD thesis, University of Massachusetts Amherst, 2001.

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

[116] G. Rens and J.-F. Raskin. Learning non-Markovian reward models in MDPs. *arXiv preprint:2001.09293*, 2020.

[117] M. Riedmiller. Neural fitted Q iteration-first experiences with a data efficient neural reinforcement learning method. In *ECML*, volume 3720, pages 317–328. Springer, 2005.

[118] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks*, pages 586–591. IEEE, 1993.

[119] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia. A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *CDC*, pages 1091–1096. IEEE, 2014.

[120] S. Safra. On the complexity of omega-automata. In *Foundations of Computer Science*, pages 319–327. IEEE, 1988.

[121] A. Salamati, A. Lavaei, S. Soudjani, and M. Zamani. Data-driven verification and synthesis of stochastic systems through barrier certificates. *arXiv preprint arXiv:2111.10330*, 2021.

[122] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[123] M. S. Santos and J. Vigo-Aguiar. Analysis of a numerical dynamic programming algorithm applied to economic models. *Econometrica*, pages 409–426, 1998.

[124] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint:1610.03295*, 2016.

[125] R. W. Shonkwiler and F. Mendivil. *Explorations in Monte Carlo Methods*. Springer Science & Business Media, 2009.

[126] S. Sickert, J. Esparza, S. Jaax, and J. Křetínskỳ. Limit-deterministic Büchi automata for linear temporal logic. In *Computer Aided Verification*, pages 312–332. Springer, 2016.

[127] S. Sickert and J. Křetínskỳ. MoChiBA: Probabilistic LTL model checking using limit-deterministic Büchi automata. In *ATVA*, pages 130–137. Springer, 2016.

[128] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[129] D. Silver, G. Lever, N. Heess, D. W. Thomas Degris, and M. Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, 2014.

[130] S. E. Z. Soudjani, C. Gevaerts, and A. Abate. FAUST^2: Formal Abstractions of Uncountable-STate STochastic Processes. In *TACAS*, pages 272–286. Springer, 2015.

[131] J. Stachurski. Continuous state dynamic programming via nonexpansive approximation. *Computational Economics*, 31(2):141–160, 2008.

[132] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT Press Cambridge, 1998.

[133] M. Svorenova, I. Cerna, and C. Belta. Optimal control of MDPs with temporal logic constraints. In *CDC*, pages 3938–3943. IEEE, 2013.

[134] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint:1801.00690*, 2018.

[135] A. L. Thomaz and C. Breazeal. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence*, 172(6-7):716–737, 2008.

[136] I. Tkachev, A. Mereacre, J.-P. Katoen, and A. Abate. Quantitative automata-based controller synthesis for non-autonomous stochastic hybrid systems. In *HSCC*, pages 293–302. ACM, 2013.

[137] I. Tkachev, A. Mereacre, J.-P. Katoen, and A. Abate. Quantitative model-checking of controlled discrete-time Markov processes. *Information and Computation*, 253:1–35, 2017.

[138] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. Teaching multiple tasks to an RL agent using LTL. In *AAMAS*, pages 452–461, 2018.

[139] M. Turchetta, F. Berkenkamp, and A. Krause. Safe exploration for interactive machine learning. *arXiv preprint:1910.13726*, 2019.

[140] M. Turchetta, A. Kolobov, S. Shah, A. Krause, and A. Agarwal. Safe reinforcement learning via curriculum induction. *arXiv preprint:2006.12136*, 2020.

[141] K. Vajjha, A. Shinnar, V. Pestun, B. Trager, and N. Fulton. CertRL: Formalizing convergence proofs for value and policy iteration in Coq. *arXiv preprint:2009.11403*, 2020.

[142] M. van Otterlo and M. Wiering. Reinforcement learning and Markov decision processes. In *Reinforcement Learning*, pages 3–42. Springer, 2012.

[143] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou, et al. Strategic attentive writer for learning macro-actions. In *NeurIPS*, pages 3486–3494, 2016.

[144] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, et al. Alphastar: Mastering the real-time strategy game StarCraft II. *DeepMind Blog*, 2019.

[145] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.

[146] Y. Wang, N. Roohi, M. West, M. Viswanathan, and G. E. Dullerud. Statistically model checking PCTL specifications on Markov decision processes via reinforcement learning. *arXiv preprint:2004.00273*, 2020.

[147] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[148] Q. Wei and X. Guo. Markov decision processes with state-dependent discount factors and unbounded rewards/costs. *Operations Research Letters*, 39(5):369–374, 2011.

[149] M. Wen, R. Ehlers, and U. Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *IROS*, pages 4983–4990. IEEE, 2015.

[150] E. M. Wolff, U. Topcu, and R. M. Murray. Robust control of uncertain Markov decision processes with temporal logic specifications. In *CDC*, pages 3372–3379. IEEE, 2012.

[151] J. Xinyu Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah. Lang2LTL: Translating natural language commands to temporal robot task specification. *arXiv preprint arXiv: 2302.11649*, 2023.

[152] R. Yan, C.-H. Cheng, and Y. Chai. Formal consistency checking over specifications in natural languages. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1677–1682. EDA Consortium, 2015.

[153] N. Yoshida, E. Uchibe, and K. Doya. Reinforcement learning with state-dependent discount factor. In *ICDL*, pages 1–6. IEEE, 2013.

[154] H. L. Younes, E. M. Clarke, and P. Zuliani. Statistical verification of probabilistic properties with unbounded until. In *Brazilian Symposium on Formal Methods*, pages 144–160. Springer, 2010.

[155] L. Z. Yuan, M. Hasanbeig, A. Abate, and D. Kroening. Modular deep reinforcement learning with temporal logic specifications. *arXiv preprint:1909.11591*, 2019.