Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees

M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, I. Lee

***Abstract*— Reinforcement Learning (RL) has emerged as an efficient method of choice for solving complex sequential decision making problems in automatic control, computer science, economics, and biology. In this paper we present a model-free RL algorithm to synthesize control policies that maximize the probability of satisfying high-level control objectives given as Linear Temporal Logic (LTL) formulas. Uncertainty is considered in the workspace properties, the structure of the workspace, and the agent actions, giving rise to a Probabilistically-Labeled Markov Decision Process (PL-MDP) with unknown graph structure and stochastic behaviour, which is even more general case than a fully unknown MDP. We first translate the LTL specification into a Limit Deterministic Büchi Automaton (LDBA), which is then used in an on-the-fly product with the PL-MDP. Thereafter, we define a synchronous reward function based on the acceptance condition of the LDBA. Finally, we show that the RL algorithm delivers a policy that maximizes the satisfaction probability asymptotically. We provide experimental results that showcase the efficiency of the proposed method.**

I. Introduction

The use of temporal logic has been promoted as formal task specifications for control synthesis in Markov Decision Processes (MDPs) due to their expressive power, as they can handle a richer class of tasks than the classical point-to-point navigation. Such rich specifications include safety and liveness requirements, sequential tasks, coverage, and temporal ordering of different objectives [1]–[5]. Control synthesis for MDPs under Linear Temporal Logic (LTL) specifications has also been studied in [6]–[10]. Common in these works is that, in order to synthesize policies that maximize the satisfaction probability, exact knowledge of the MDP is required. Specifically, these methods construct a product MDP by composing the MDP that captures the underlying dynamics with a Deterministic Rabin Automaton (DRA) that represents the LTL specification. Then, given the product MDP, probabilistic model checking techniques are employed to design optimal control policies [11], [12].

In this paper, we address the problem of designing optimal control policies for MDPs with *unknown* stochastic behaviour so that the generated traces satisfy a given LTL specification with maximum probability. Unlike previous work, uncertainty is considered both in the environment properties and in the agent actions, provoking a *Probabilistically-Labeled* MDP (PL-MDP). This model further extend MDPs to provide a way to consider dynamic and uncertain environments. In order to solve this problem, we first convert the LTL formula into a Limit Deterministic Büchi Automaton (LDBA) [13]. It is known that this construction results in an exponential-sized automaton for $\text{LTL}_{\setminus \text{GU}}$, and it results in nearly the same size as a DRA for the rest of LTL. $\text{LTL}_{\setminus \text{GU}}$ is a fragment of linear temporal logic with the restriction that no until operator occurs in the scope of an always operator. On the other hand, the DRA that are typically employed in relevant work are doubly exponential in the size of the original LTL formula [14]. Furthermore, a Büchi automaton is semantically simpler than a Rabin automaton in terms of its acceptance conditions [10], [15], which makes our algorithm much easier to implement. Once the LDBA is generated from the given LTL property, we construct on-the-fly a product between the PL-MDP and the resulting LDBA and then define a *synchronous reward* function based on the acceptance condition of the Büchi automaton over the state-action pairs of the product. Using this algorithmic reward shaping procedure, a model-free RL algorithm is introduced, which is able to generate a policy that returns the maximum expected reward. Finally, we show that maximizing the expected accumulated reward entails the maximization of the satisfaction probability.

Related work – A model-based RL algorithm to design policies that maximize the satisfaction probability is proposed in [16], [17]. Specifically, [16] assumes that the given MDP model has unknown transition probabilities and builds a Probably Approximately Correct MDP (PAC MDP), which is composed with the DRA that expresses the LTL property. The overall goal is to calculate the 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 PAC MDP is generated via an RL-like algorithm, then value iteration is applied to update state values. A similar model-based solution is proposed in [18]: this also hinges on approximating the transition probabilities, which limits the precision of the policy generation process. Unlike the problem that is considered in this paper, the work in [18] is limited to policies whose traces satisfy the property with probability one. Moreover, [16]–[18] require to learn all transition probabilities of the MDP. As a result, they need a significant amount of memory to store the learned model [19]. This specific issue is addressed in [20], which proposes an actor-critic method for LTL specification that requires the graph structure of the MDP, but not all transition probabilities. The structure of the MDP allows for the computation of

M. Hasanbeig, A. Abate, and D. Kroening are with the Department of Computer Science, University of Oxford, UK, and are supported by the ERC project 280053 (CPROVER) and the H2020 FET OPEN 712689 SC^2. {hosein.hasanbeig,aabate,kroening}@cs.ox.ac.uk. Y. Kantaros, G. J. Pappas, and I. Lee are with the School of Engineering and Applied Science (SEAS), University of Pennsylvania, PA, USA, and are supported by the AFRL and DARPA under Contract No. FA8750-18-C-0090 and the ARL RCTA under Contract No. W911NF-10-2-0016 {kantaros,pappasg,lee}@seas.upenn.edu.

Accepting Maximum End Components (AMECs) in the product MDP, while transition probabilities are generated only when needed by a simulator. By contrast, the proposed method does not require knowledge of the structure of the MDP and does not rely on computing AMECs of a product MDP. A model-free and AMEC-free RL algorithm for LTL planning is also proposed in [21]. Nevertheless, unlike our proposed method, all these cognate contributions rely on the LTL-to-DRA conversion, and uncertainty is considered only in the agent actions, but not in the workspace properties.

In [22] and [23] safety-critical settings in RL are addressed in which the agent has to deal with a heterogeneous set of MDPs in the context of cyber-physical systems. [24] further employs DDL [25], a first-order multi-modal logic for specifying and proving properties of hybrid programs.

The first use of LDBA for LTL-constrained policy synthesis in a model-free RL setup appears in [26], [27]. Specifically, [27] propose a hybrid neural network architecture combined with LDBAs to handle MDPs with continuous state spaces. The work in [26] has been taken up more recently by [28], which has focused on model-free aspects of the algorithm and has employed a different LDBA structure and reward, which introduce extra states in the product MDP. The authors also do not discuss the complexity of the automaton construction with respect to the size of the formula, but given the fact that resulting automaton is not a generalised Büchi, it can be expected that the density of automaton acceptance condition is quite low, which might result in a state-space explosion, particularly if the LTL formula is complex. As we show in the proof for the counter example in the Appendix-E the authors indeed have overlooked that our algorithm is episodic, and allows the discount factor to be equal to one. Unlike [26]–[28], in this work we consider uncertainty in the workspace properties by employing PL-MDPs.

Summary of contributions – *First*, we propose a model-free RL algorithm to synthesize control policies for *unknown* PL-MDPs which maximizes the probability of satisfying LTL specifications. *Second*, we define a *synchronous reward function* and we show that maximizing the accumulated reward maximizes the satisfaction probability. *Third*, we convert the LTL specification into an LDBA which, as a result, shrinks the state-space that needs to explored compared to relevant LTL-to-DRA-based works in finite-state MDPs. Moreover, unlike previous works, our proposed method does not require computation of AMECs of a product MDP, which avoids the quadratic time complexity of such a computation in the size of the product MDP [11], [12].

II. Problem Formulation

Consider a robot that resides in a partitioned environment with a finite number of states. To capture uncertainty in both the robot motion and the workspace properties, we model the interaction of the robot with the environment as a *PL-MDP*, which is defined as follows.

Definition 2.1 (Probabilistically-Labeled MDP [9]): A PL-MDP is a tuple $\mathfrak{M} = (\mathcal{X}, x_0, \mathcal{A}, P_C, \mathcal{AP}, P_L)$, where $\mathcal{X}$ is a finite set of states; $x_0 \in \mathcal{X}$ is the initial state; $\mathcal{A}$ is a finite set of actions. With slight abuse of notation $\mathcal{A}(x)$ denotes the available actions at state $x \in \mathcal{X}$; $P_C : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to [0,1]$ is the transition probability function so that $P_C(x,a,x')$ is the transition probability from state $x \in \mathcal{X}$ to state $x' \in \mathcal{X}$ via control action $a \in \mathcal{A}$ and $\sum_{x' \in \mathcal{S}} P_C(x,a,x') = 1$, for all $a \in \mathcal{A}(x)$; $\mathcal{AP}$ is a set of atomic propositions; and $P_L : \mathcal{X} \times 2^{\mathcal{AP}} \to [0,1]$ specifies the associated probability. Specifically, $P_L(x,\ell)$ denotes the probability that $\ell \in 2^{\mathcal{AP}}$ is observed at state $x \in \mathcal{X}$, where $\sum_{\ell \in 2^{\mathcal{AP}}} P_L(x,\ell) = 1$, $\forall x \in \mathcal{X}$. □

The probabilistic map P_L provides a means to model dynamic and uncertain environments. Hereafter, we assume that the PL-MDP $\mathfrak{M}$ is fully observable, i.e., at any time/stage t the current state, denoted by x^t, and the observations in state x^t, denoted by $\ell^t \in 2^{\mathcal{AP}}$, are known.

At any stage $T \geq 0$ we define the robot's past path as $X_T = x_0 x_1 \dots x_T$, the past sequence of observed labels as $L_T = \ell_0 \ell_1 \dots \ell_T$, where $\ell_t \in 2^{\mathcal{AP}}$ and the past sequence of control actions $\mathcal{A}_T = a_0 a_1 \dots a_{T-1}$, where $a_t \in \mathcal{A}(x_t)$. These three sequences can be composed into a complete past run, defined as $R_T = x_0 \ell_0 a_0 x_1 \ell_1 a_1 \dots x_T \ell_T$. We denote by $\mathcal{X}_T$, $\mathcal{L}_T$, and $\mathcal{R}_T$ the set of all possible sequences X_T, L_T and R_T, respectively.

The goal of the robot is accomplish a task expressed as an LTL formula. LTL is a formal language that comprises a set of atomic propositions $\mathcal{AP}$, the Boolean operators, i.e., conjunction $\wedge$ and negation $\neg$, and two temporal operators, next $\bigcirc$ and until $\cup$. LTL formulas over a set $\mathcal{AP}$ can be constructed based on the following grammar:

$$\phi ::= \text{true} \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \ \cup \ \phi_2,$$

where $\pi \in \mathcal{AP}$. The other Boolean and temporal operators, e.g., *always* $\square$, have their standard syntax and meaning. An infinite *word* σ over the alphabet $2^{\mathcal{AP}}$ is defined as an infinite sequence $\sigma = \pi_0 \pi_1 \pi_2 \dots \in (2^{\mathcal{AP}})^\omega$, where ω denotes infinite repetition and $\pi_k \in 2^{\mathcal{AP}}$, $\forall k \in \mathbb{N}$. The language $\{\sigma \in (2^{\mathcal{AP}})^\omega | \sigma \models \phi\}$ is defined as the set of words that satisfy the LTL formula ϕ, where $\models \subseteq (2^{\mathcal{AP}})^\omega \times \phi$ is the satisfaction relation [29].

In what follows, we define the probability that a stationary policy for $\mathfrak{M}$ satisfies the assigned LTL specification. Specifically, a stationary policy $\boldsymbol{\xi}$ for $\mathfrak{M}$ is defined as $\boldsymbol{\xi} = \xi_0 \xi_1 \dots$, where $\xi_t : \mathcal{X} \times \mathcal{A} \to [0,1]$. Given a stationary policy $\boldsymbol{\xi}$, the probability measure $\mathbb{P}^{\boldsymbol{\xi}}_{\mathfrak{M}}$, defined on the smallest σ-algebra over $\mathcal{R}_\infty$, is the unique measure defined as $\mathbb{P}^{\boldsymbol{\xi}}_{\mathfrak{M}} = \prod_{t=0}^{T} P_C(x_t, a_t, x_{t+1}) P_L(x_t, \ell_t) \xi_t(x_t, a_t)$, where $\xi_t(x_t, a_t)$ denotes the probability that at time t the action a_t will be selected given the current state x_t [11], [30]. We then define the probability of $\mathfrak{M}$ satisfying ϕ under policy $\boldsymbol{\xi}$ as [11], [12]

$$\mathbb{P}^{\boldsymbol{\xi}}_{\mathfrak{M}}(\phi) = \mathbb{P}^{\boldsymbol{\xi}}_{\mathfrak{M}}(\mathcal{R}_\infty : \mathcal{L}_\infty \models \phi), \tag{1}$$

The problem we address in this paper is summarized as follows.

Problem 1: Given a PL-MDP $\mathfrak{M}$ with unknown transition probabilities, unknown label mapping, unknown underlying graph structure, and a task specification captured by an LTL

formula ϕ, synthesize a deterministic stationary control policy $\boldsymbol{\xi}^*$ that maximizes the probability of satisfying ϕ captured in (1), i.e., $\boldsymbol{\xi}^* = \text{argmax}_{\boldsymbol{\xi}} \mathbb{P}^{\boldsymbol{\xi}}_{\mathfrak{M}}(\phi)$.[1] □

III. A New Learning-for-Planning Algorithm

In this section, we first discuss how to translate the LTL formula into an LDBA $\mathfrak{A}$ (see Section III-A). Then, we define the product MDP $\mathfrak{P}$, constructed by composing the PL-MDP $\mathfrak{M}$ and the LDBA $\mathfrak{A}$ that expresses ϕ (see Section III-B). Next, we assign rewards to the product MDP transitions based on the accepting condition of the LDBA $\mathfrak{A}$. As we show later, this allows us to synthesize a policy $\boldsymbol{\mu}^*$ for $\mathfrak{P}$ that maximizes the probability of satisfying the acceptance conditions of the LDBA. The projection of the obtained policy $\boldsymbol{\mu}^*$ over model $\mathfrak{M}$ results in a policy $\boldsymbol{\xi}^*$ that solves Problem 1 (Section III-C).

A. Translating LTL into an LDBA

An LTL formula ϕ can be translated into an automaton, namely a finite-state machine that can express the set of words that satisfy ϕ. Conventional probabilistic model checking methods translate LTL specifications into DRAs, which are then composed with the PL-MDP, giving rise to a product MDP. Nevertheless, it is known that this conversion results, in the worst case, in automata that are doubly exponential in the size of the original LTL formula [14]. By contrast, in this paper we propose to express the given LTL property as an LDBA, which results in a much more succinct automaton [13], [15]. This is the key to the *reduction of the state-space that needs to be explored*; see also Section V.

Before defining the LDBA, we first need to define the Generalized Büchi Automaton (GBA).

Definition 3.1 (Generalized Büchi Automaton [11]): 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, $\mathcal{F} = \mathcal{F}_1, \ldots, \mathcal{F}_f$ is the set of accepting conditions where $\mathcal{F}_j \subset \mathcal{Q}$, $1 \leq j \leq f$, and $\delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$ is a transition relation. □

An infinite run ρ of $\mathfrak{A}$ over an infinite word $\sigma = \pi_0\pi_1\pi_2 \cdots \in \Sigma^\omega$, $\pi_k \in \Sigma = 2^{\mathcal{AP}}$ $\forall k \in \mathbb{N}$, is an infinite sequence of states $q_k \in \mathcal{Q}$, i.e., $\rho = q_0 q_1 \ldots q_k \ldots$, such that $q_{k+1} \in \delta(q_k, \pi_k)$. The infinite run ρ is called *accepting* (and the respective word σ is accepted by the GBA) if $\texttt{Inf}(\rho) \cap \mathcal{F}_j \neq \emptyset, \forall j \in \{1, \ldots, f\}$, where $\texttt{Inf}(\rho)$ is the set of states that are visited infinitely often by ρ.

Definition 3.2 (Limit Deterministic Büchi Automaton [13]): 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 $\mathcal{Q} = \mathcal{Q}_N \cup \mathcal{Q}_D$, so that (i) $\delta(q, \pi) \subset \mathcal{Q}_D$ and $|\delta(q, \pi)| = 1$, for every state $q \in \mathcal{Q}_D$ and $\pi \in \Sigma$; and (ii) for every $\mathcal{F}_j \in \mathcal{F}$, it holds that $\mathcal{F}_j \subset \mathcal{Q}_D$ and there are ε-transitions from $\mathcal{Q}_N$ to $\mathcal{Q}_D$. □

An ε-transition allows the automaton to change its state without reading any specific input. In practice, the ε-transitions between $\mathcal{Q}_N$ and $\mathcal{Q}_D$ reflect the "guess" on reaching $\mathcal{Q}_D$: accordingly, if after an ε-transition the associated labels in the accepting set of the automaton cannot be read, or if the accepting states cannot be visited, then the guess is deemed to be wrong, and the trace is disregarded and is not accepted by the automaton. However, if the trace is accepting, then the trace will stay in $\mathcal{Q}_D$ ever after, i.e. $\mathcal{Q}_D$ is invariant.

Definition 3.3 (Non-accepting Sink Component): A non-accepting sink component in an LDBA $\mathfrak{A}$ is a directed graph induced by a set of states $\mathcal{Q}_{sink} \subset \mathcal{Q}$ such that (1) is strongly connected, (2) does not include all accepting sets $\mathcal{F}_j$, $j = 1, ..., f$, and (3) there exist no other strongly connected set $\mathcal{Q}' \subset \mathcal{Q}$, $\mathcal{Q}' \neq \mathcal{Q}_{sink}$ that $\mathcal{Q}_{sink} \subset \mathcal{Q}'$. We denote the union set of all non-accepting sink components as $\mathcal{Q}_{sinks}$. □

B. Product MDP

Given the PL-MDP $\mathfrak{M}$ and the LDBA $\mathfrak{A}$, we define the product MDP $\mathfrak{P} = \mathfrak{M} \times \mathfrak{A}$ as follows.

Definition 3.4 (Product MDP): Given a PL-MDP $\mathfrak{M} = (\mathcal{X}, x_0, \mathcal{A}, P_C, \mathcal{AP}, P_L)$ and an LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \delta)$, we define the product MDP $\mathfrak{P} = \mathfrak{M} \times \mathfrak{A}$ as $\mathfrak{P} = (\mathcal{S}, s_0, \mathcal{A}_\mathfrak{P}, P_\mathfrak{P}, \mathcal{F}_\mathfrak{P})$, where (i) $\mathcal{S} = \mathcal{X} \times 2^{\mathcal{AP}} \times \mathcal{Q}$ is the set of states, so that $s = (x, \ell, q) \in \mathcal{S}$, $x \in \mathcal{X}$, $\ell \in 2^{\mathcal{AP}}$, and $q \in \mathcal{Q}$; (ii) $s_0 = (x_0, \ell_0, q_0)$ is the initial state; (iii) $\mathcal{A}_\mathfrak{P}$ is the set of actions inherited from the MDP, so that $\mathcal{A}_\mathfrak{P}(s) = \mathcal{A}(x)$, where $s = (x, \ell, q)$; (iv) $P_\mathfrak{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} : [0, 1]$ is the transition probability function, so that

$$P_\mathfrak{P}([x, \ell, q], a, [x', \ell', q']) = P_C(x, u, x') P_L(x', \ell'), \quad (2)$$

where $[x, \ell, q] \in \mathcal{S}$, $[x', \ell', q'] \in \mathcal{S}$, $a \in \mathcal{A}(x)$ and $q' = \delta(q, \ell')$; (v) $\mathcal{F}_\mathfrak{P} = \{(\mathcal{F}^\mathfrak{P}_j), j = 1, \ldots, f\}$ is the set of accepting states, where $\mathcal{F}^\mathfrak{P}_j = \mathcal{X} \times 2^{\mathcal{AP}} \times \mathcal{F}_j$. In order to handle ε-transitions in the constructed LDBA we have to add the following modifications to the standard definition of the product MDP [15]. First, for every ε-transition to a state $q' \in \mathcal{Q}$ we add an action $\varepsilon_{q'}$ in the product MDP, i.e., $\mathcal{A}_\mathfrak{P}(s) = \mathcal{A}_\mathfrak{P}(s) \cup \{\varepsilon_{q'}, s' = [x, \ell', q'], q' \in \mathcal{Q}\}$. Second, the transition probabilities of ε-transitions are given by

$$P_\mathfrak{P}(s, a, s') = \begin{cases} 1, & \text{if } (x = x') \wedge (\ell = \ell') \wedge (\delta(q, \varepsilon_{q'}) = q') \\ 0, & \text{otherwise,} \end{cases} \quad (3)$$

where $s = (x, \ell, q)$ and $s' = (x', \ell', q')$. □

Given any policy $\boldsymbol{\mu}$ for $\mathfrak{P}$, we define an infinite run $\rho^{\boldsymbol{\mu}}_\mathfrak{P}$ of $\mathfrak{P}$ to be an infinite sequence of states of $\mathfrak{P}$, i.e., $\rho^{\boldsymbol{\mu}}_\mathfrak{P} = s_0 s_1 s_2 \ldots$, where $P_\mathfrak{P}(s_t, \boldsymbol{\mu}(s_t), s_{t+1}) > 0$. By definition of the accepting condition of the LDBA $\mathfrak{A}$, an infinite run $\rho^{\boldsymbol{\mu}}_\mathfrak{P}$ is accepting, i.e., $\boldsymbol{\mu}$ satisfies ϕ with a non-zero probability (denoted by $\boldsymbol{\mu} \models \phi$), if $\texttt{Inf}(\rho^{\boldsymbol{\mu}}_\mathfrak{P}) \cap \mathcal{F}^\mathfrak{P}_j \neq \emptyset$, $\forall j \in \{1, \ldots, f\}$.

In what follows, we design a synchronous reward function based on the accepting condition of the LDBA so that maximization of the expected accumulated reward implies maximization of the satisfaction probability. Specifically, we generate a control policy $\boldsymbol{\mu}^*$ that maximizes the probability of (i) reaching the states of $\mathcal{F}_\mathfrak{P}$ from s_0 and (ii) the probability that each accepting set $\mathcal{F}^\mathfrak{P}_j$ will be visited infinitely often.

[1] The fact that the graph structure is unknown implies that we do not know which transition probabilities are equal to zero. As a result, relevant approaches that require the structure of the MDP, as e.g., [20] cannot be applied.

C. Construction of the Reward Function

To synthesize a policy that maximizes the probability of satisfying ϕ, we construct a synchronous reward function for the product MDP. The main idea is that (i) visiting a set $\mathcal{F}_j$, $1 \leq j \leq f$ yields a positive reward $r > 0$; and (ii) revisiting the same set $\mathcal{F}_j$ returns zero reward until all other sets $\mathcal{F}_k$, $k \neq j$ are also visited; (iii) the rest of the transitions have zero rewards. Intuitively, this reward shaping strategy motivates the agent to visit *all* accepting sets $\mathcal{F}_j$ of the LDBA infinitely often, as required by the acceptance condition of the LDBA; see also Section IV.

To formally present the proposed reward shaping method, we need first to introduce the the *accepting frontier set* $\mathbb{A}$ which is initialized as the family set

$$\mathbb{A} = \{F_k\}_{k=1}^{f}. \tag{4}$$

This set is updated on-the-fly every time a set $\mathcal{F}_j$ is visited as $\mathbb{A} \leftarrow AF(q, \mathbb{A})$ where $AF(q, \mathbb{A})$ is the *accepting frontier function* defined as follows.

Definition 3.5 (Accepting Frontier Function): Given an LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \delta)$, we define $AF : \mathcal{Q} \times 2^{\mathcal{Q}} \rightarrow 2^{\mathcal{Q}}$ as the accepting frontier function, which executes the following operation over any given set $\mathbb{A} \in 2^{\mathcal{Q}}$:

$$AF(q, \mathbb{A}) = \begin{cases} \mathbb{A} \setminus \mathcal{F}_j & : \ (q \in \mathcal{F}_j) \wedge (\mathbb{A} \neq \mathcal{F}_j) \\ & : \\ \{F_k\}_{k=1}^{f} \setminus \mathcal{F}_j & : \ (q \in \mathcal{F}_j) \wedge (\mathbb{A} = \mathcal{F}_j). \end{cases} \quad \square$$

In words, given a state $q \in \mathcal{F}_j$ and the set $\mathbb{A}$, AF outputs a set containing the elements of $\mathbb{A}$ minus those elements that are common with $\mathcal{F}_j$ (first case). However, if $\mathbb{A} = \mathcal{F}_j$, then the output is the family set of all accepting sets of $\mathfrak{A}$ minus those elements that are common with $\mathcal{F}_j$, resulting in a reset of $\mathbb{A}$ to (4) minus those elements that are common with $\mathcal{F}_j$ (second case). Intuitively, $\mathbb{A}$ always contains those accepting sets that are needed to be visited at a given time and in this sense the reward function is synchronous with the LDBA accepting condition.

Given the accepting frontier set $\mathbb{A}$, we define the following reward function

$$R(s, a) = \begin{cases} r & \text{if } q' \in \mathbb{A}, \ s' = (x', \ell', q'), \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

In (5), s' is the state of the product MDP that is reached from state s by taking action a, and $r > 0$ is an arbitrary positive reward. In this way the agent is guided to visit all accepting sets $\mathcal{F}_j$ infinitely often and, consequently, satisfy the given LTL property.

Remark 3.6: The initial and accepting components of the LDBA proposed in [13] (as used in this paper) are both deterministic. By Definition 3.2, the discussed LDBA is indeed a limit-deterministic automaton, however notice that the obtained determinism within its initial part is stronger than that required in the definition of LDBA. Thanks to this feature of the LDBA structure, in our proposed algorithm there is no need to "explicitly build" the product MDP and to store all its states in memory. The automaton transitions can be executed on-the-fly, as the agent reads the labels of the MDP states. $\square$

Given $\mathfrak{P}$, we compute a *stationary deterministic* policy $\boldsymbol{\mu}^*$, that maximizes the expected accumulated return, i.e.,

$$\boldsymbol{\mu}^*(s) = \arg\max_{\boldsymbol{\mu} \in \mathcal{D}} \ U^{\boldsymbol{\mu}}(s), \tag{6}$$

where $\mathcal{D}$ is the set of all stationary deterministic policies over $\mathcal{S}$, and

$$U^{\boldsymbol{\mu}}(s) = \mathbb{E}^{\boldsymbol{\mu}}[\sum_{n=0}^{\infty} \gamma^n \ R(s_n, \boldsymbol{\mu}(s_n)) | s_0 = s], \tag{7}$$

where $\mathbb{E}^{\boldsymbol{\mu}}[\cdot]$ denotes the expected value given that the product MDP follows the policy $\boldsymbol{\mu}$ [30], $0 \leq \gamma \leq 1$ is the discount factor, and $s_0, ..., s_n$ is the sequence of states generated by policy $\boldsymbol{\mu}$ up to time step n, initialized at $s_0 = s$. Note that the optimal policy is stationary as shown in the following result.

Theorem 3.7 ([30]): In any finite-state MDP, such as $\mathfrak{P}$, if there exists an optimal policy, then that policy is stationary and deterministic. $\square$

In order to construct $\boldsymbol{\mu}^*$, we employ episodic Q-learning (QL), a model-free RL scheme described in Algorithm 1.[2] Specifically, Algorithm 1 requires as inputs (i) the LDBA $\mathfrak{A}$, (ii) the reward function R defined in (5), and (iii) the hyper-parameters of the learning algorithm.

Observe that in Algorithm 1, we use an action-value function $Q : \mathcal{S} \times \mathcal{A}_{\mathfrak{P}} \rightarrow \mathbb{R}$ to evaluate $\boldsymbol{\mu}$ instead of $U^{\boldsymbol{\mu}}(s)$, since the MDP $\mathfrak{P}$ is unknown. The action-value function $Q(s, a)$ can be initialized arbitrarily. Note that $U^{\boldsymbol{\mu}}(s) = \max_{a \in \mathcal{A}_{\mathfrak{P}}} Q(s, a)$. Also, we define a function $C : \mathcal{S} \times \mathcal{A}_{\mathfrak{P}} \rightarrow \mathbb{N}$ that counts the number of times that action a has been taken at state s. The policy $\boldsymbol{\mu}$ is selected to be an ϵ-greedy policy, which means that with probability $1-\epsilon$, the greedy action $\text{argmax}_{a \in \mathcal{A}_{\mathfrak{P}}} Q(s, a)$ is taken, and with probability ϵ a random action a is selected. Every episode terminates when the current state of the automaton gets inside $\mathcal{Q}_{sinks}$ (Definition 3.3) or when the iteration number in the episode reaches a certain threshold τ. Note that it holds that $\boldsymbol{\mu}$ asymptotically converges to the optimal greedy policy $\boldsymbol{\mu}^* = \text{argmax}_{a \in \mathcal{A}_{\mathfrak{P}}} Q^*(s, a)$: where Q^* is the optimal Q function. Further, $Q(s, \boldsymbol{\mu}^*(s)) = U^{\boldsymbol{\mu}^*}(s) = V^*(s)$, where $V^*(s)$ is the optimal value function that could have been computed via Dynamic Programming (DP) if the MDP was fully known [19], [31], [32]. Projection of $\boldsymbol{\mu}^*$ onto the state-space of the PL-MDP, yields the finite-memory policy $\boldsymbol{\xi}^*$ that solves Problem 1.

IV. Analysis of the Algorithm

In this section, we show that the policy $\boldsymbol{\mu}^*$ generated by Algorithm 1 maximizes (1), i.e., the probability of satisfying the property ϕ. Furthermore, we show that, unlike existing approaches, our algorithm can produce the best available

[2]Note that any other off-the-shelf model-free RL algorithm can also be used within Algorithm 1, including any variant of the class of temporal difference learning algorithms [19].

Algorithm 1: RL for LTL objective

input : Reward function R, LDBA $\mathfrak{A}$,γ,τ
output : $\boldsymbol{\mu}^*$

Initialize $C(s,a)=0$, $Q(s,a)$, $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}_{\mathfrak{P}}$
  $\mathbb{A} = \bigcup_{k=1}^{f} F_k$
  $episode\text{-}number := 0$, $iteration\text{-}number := 0$
while Q *is not converged* **do**
  $episode\text{-}number++$
  $s_{\text{cur}} = s_0$
  $\epsilon = 1/(episode\text{-}number)$
  while $(q \notin \mathcal{Q}_{sinks}) \wedge (iteration\text{-}number < \tau)$ **do**
    $iteration\text{-}number++$
    Set $a_{\text{cur}} = \text{argmax}_{a \in \mathcal{A}_{\mathfrak{P}}} Q(s,a)$ with probability $1-\epsilon$ and set a_{cur} as a random action in $\mathcal{A}_{\mathfrak{P}}$ with probability ϵ
    Execute a_{cur} and observe $s_{\text{next}} = (x_{next}, \ell_{next}, q_{next})$, and $R(s_{\text{cur}}, a_{\text{cur}})$
    if $R(s_{cur}, a_{cur}) > 0$ **then**
      $\mathbb{A} = AF(q_{\text{next}}, \mathbb{A})$,
    $C(s_{\text{cur}}, a_{\text{cur}})++$
    $Q(s_{\text{cur}}, a_{\text{cur}}) =$
    $Q(s_{\text{cur}}, a_{\text{cur}}) + (1/C(s_{\text{cur}}, a_{\text{cur}}))[R(s_{\text{cur}}, a_{\text{cur}}) -$
    $Q(s_{\text{cur}}, a_{\text{cur}}) + \gamma \max_{a'}(s_{\text{next}}, a'))]$
    $s_{\text{cur}} = s_{\text{next}}$
  end
end

policy if the property cannot be satisfied. To prove these claims, we need to show the following results. All proofs are presented in the Appendix. First, we show that the accepting frontier set $\mathbb{A}$ is time-invariant. This is needed to ensure that the LTL formula is satisfied over the product MDP by a stationary policy.

Proposition 4.1: For an LTL formula ϕ and its associated LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \delta)$, the accepting frontier set $\mathbb{A}$ is time-invariant at each state of $\mathfrak{A}$. □

As stated earlier, since QL is proved to converge to the optimal Q-function [19], it can synthesize an optimal policy with respect to the given reward function. The following result shows that the optimal policy produced by Algorithm 1 satisfies the given LTL property.

Theorem 4.2: Assume that there exists at least one deterministic stationary policy in $\mathfrak{P}$ whose traces satisfy the property ϕ with positive probability. Then the traces of the optimal policy $\boldsymbol{\mu}^*$ defined in (6) satisfy ϕ with positive probability, as well.

Next we show that $\boldsymbol{\mu}^*$ and subsequently its projection $\boldsymbol{\xi}^*$ maximize the satisfaction probability.

Theorem 4.3: If an LTL property ϕ is satisfiable by the PL-MDP $\mathfrak{M}$, then the optimal policy $\boldsymbol{\mu}^*$ that maximizes the expected accumulated reward, as defined in (6), maximizes the probability of satisfying ϕ, defined in (1), as well. □

Next, we show that if there does not exist a policy that satisfies the LTL property ϕ, Algorithm 1 will find the policy that is the closest one to property satisfaction. To this end, we first introduce the notion of *closeness to satisfaction*.

Definition 4.4 (Closeness to Satisfaction): Assume that two policies $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ do not satisfy the property ϕ. Consequently, there are accepting sets in the automaton

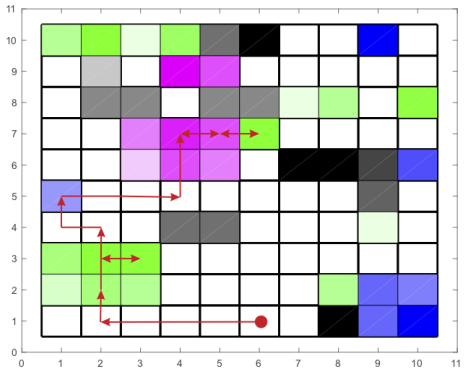


Fig. 1: PL-MDP that models the interaction of the robot with the environment. The color of each region (square) corresponds to the probability that some event can be observed there. Specifically, gray, magenta, blue, and green mean that there is a non-zero probability of an obstacle (obs), a user (user), target 1 (target1), and target 2 target2. Higher intensity indicates a higher probability. The red trajectory represents a sample path of the robot with the optimal control strategy $\boldsymbol{\xi}^*$ for the first case study. The red dot is the initial location of the robot.

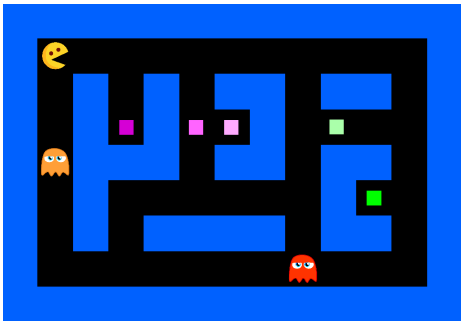


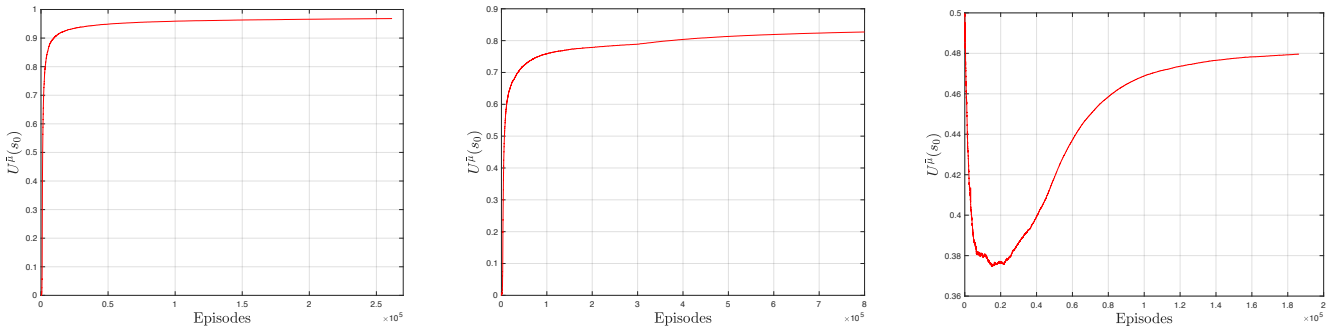
Fig. 2: Initial condition in Pacman environment. The magenta square is labeled food1 and the green one food2. The color intensity of each square corresponds to the probability of the food being observed. The state of being caught by a ghost is labeled ghost and the rest of the state space neutral.

that have no intersection with runs of the induced Markov chains $\mathfrak{P}^{\boldsymbol{\mu}_1}$ and $\mathfrak{P}^{\boldsymbol{\mu}_2}$. The policy $\boldsymbol{\mu}_1$ is closer to satisfying the property if runs of $\mathfrak{P}^{\boldsymbol{\mu}_1}$ have more intersections with accepting sets of the automaton than runs of $\mathfrak{P}^{\boldsymbol{\mu}_2}$. □

Corollary 4.5: If there does not exist a policy in the PL-MDP $\mathfrak{M}$ that satisfies the property ϕ, then proposed algorithm yields a policy that is closest to satisfying ϕ. □

V. Experiments

In this section we present three case studies, implemented on MATLAB R2016a on a computer with an Intel Xeon CPU at 2.93 GHz and 4 GB RAM. In the first two experiments, the environment is represented as a 10×10 discrete grid world, as illustrated in Figure 1. The third case study is an adaptation of the well-known Atari game Pacman (Figure 2), which is initialized in a configuration that is quite hard for the agent to solve.



(a) Case Study I (b) Case Study II (c) Case Study III

Fig. 3: Illustration of the evolution of $U^{\bar{\boldsymbol{\mu}}}(s_0)$ with respect to episodes. $\bar{\boldsymbol{\mu}}$ denotes the ϵ-greedy policy which converges to the optimal greedy policy $\boldsymbol{\mu}^*$. Videos of Pacman winning the game can be found in [33]

The first case study pertains to a temporal logic planning problem in a dynamic and unknown environment with AMECs, while the second one does not admit AMECs. Note that the majority of existing algorithms fail to provide a control policy when AMECs do not exist [8], [20], [34], or result in control policies without satisfaction guarantees [18].

The LTL formula considered in the first two case studies is the following:

$$\begin{aligned}\phi_1 =& \Diamond(\text{target1}) \wedge \square\Diamond(\text{target2}) \wedge \square\Diamond(\text{user}) \wedge \\ & (\neg\text{user} \cup \text{target2}) \wedge \square(\neg\text{obs}).\end{aligned} \tag{8}$$

In words, this LTL formula requires the robot to (i) eventually visit target 1 (expressed as $\Diamond$target1); (ii) visit target 2 infinitely often and take a picture of it ($\square\Diamond$target2); (iii) visit a user infinitely often where, say, the collected pictures are uploaded (captured by $\square\Diamond$user); (iv) avoid visiting the user until a picture of target 2 has been taken; and (v) always avoid obstacles (captured by $\square(\neg$obs).

The LTL formula (8) can be expressed as a DRA with 11 states. On the other hand, a corresponding LDBA has 5 states (fewer, as expected), which results in a significant reduction of the state space that needs to be explored.

The interaction of the robot with the environment is modeled by a PL-MDP $\mathfrak{M}$ with 100 states and 10 actions per state. The actions space is $\{Up, Right, Down, Left, None\} \times \{Take\ picture, Do\ not\ take\ picture\}$. We assume that the targets and the user are dynamic, i.e., their location in the environment varies probabilistically. Specifically, their presence in a given region $x \in \mathcal{X}$ is determined by the unknown function P_L from Definition 2.1 (Figure 1).

The LTL formula specifying the task for Pacman (third case study) is:

$$\begin{aligned}\phi_2 =& \Diamond[(\text{food1} \wedge \Diamond\text{food2}) \vee (\text{food2} \wedge \Diamond\text{food1})] \wedge \\ & \square(\neg\text{ghost}).\end{aligned} \tag{9}$$

Intuitively, the agent is tasked with (i) eventually eating food1 and then food2 (or vice versa), while (ii) avoiding any contact with the ghosts. This LTL formula corresponds to a DRA with 5 states and to an LDBA with 4 states. The agent can execute 5 actions per state $\{Up, Right, Down, Left, None\}$ and if the agent hits a wall by taking an action it remains in the previous location. The ghosts dynamics are stochastic: with a probability $p_g = 0.9$ each ghost chases the Pacman (often referred to as "chase mode"), and with its complement it executes a random action ("scatter mode").

In the first case study, we assume that there is no uncertainty in the robot actions. In this case, it can be verified that AMECs exist. Figure 3(a) illustrates the evolution of $U^{\bar{\boldsymbol{\mu}}}(s_0)$ over 260000 episodes, where $\bar{\boldsymbol{\mu}}$ denotes the ϵ-greedy policy. The optimal policy was constructed in approximately 30 minutes. A sample path of the robot with the projection of optimal control strategy $\boldsymbol{\mu}^*$ onto $\mathcal{X}$, i.e. policy $\boldsymbol{\xi}^*$, is given in Figure 1 (red path).

In the second case study, we assume that the robot is equipped with a noisy controller and, therefore, it can execute the desired action with probability 0.8, whereas a random action among the other available ones is taken with a probability of 0.2. In this case, it can be verified that AMECs do not exist. Intuitively, the reason why AMECs do not exist is that there is always a non-zero probability with which the robot will hit an obstacle while it travels between the access point and target 2 and, therefore, it will violate ϕ. Figure 3(b) shows the evolution of $U^{\bar{\boldsymbol{\mu}}}(s_0)$ over 800000 episodes for the ϵ-greedy policy. The optimal policy was synthesized in approximately 2 hours.

In the third experiment, there is no uncertainty in the execution of actions, namely the motion of the Pacman agent is deterministic. Figure 3(c) shows the evolution of $U^{\bar{\boldsymbol{\mu}}}(s_0)$ over 186000 episodes where $\bar{\boldsymbol{\mu}}$ denotes the ϵ-greedy policy. On the other hand, the use of standard Q-learning (without LTL guidance) would require either to construct a history-dependent reward for the PL-MDP $\mathfrak{M}$ as a proxy for the considered LTL property, which is very challenging for complex LTL formulas, or to perform exhaustive state-space search with static rewards, which is evidently quite wasteful and failed to generate an optimal policy in our experiments.

Note that given the policy $\boldsymbol{\xi}^*$ for the PL-MDP, probabilistic model checkers, such as PRISM [35], or standard Dynamic Programming methods can be employed to compute the probability of satisfying ϕ. For instance, for the first case

study, the synthesized policy satisfies ϕ with probability 1, while for the second case study, the satisfaction probability is 0, since AMECs do not exist. For the same reason, even if the transition probabilities of the PL-MDP are known, PRISM could not generate a policy for the second case study. Nevertheless, the proposed algorithm can synthesize the closest-to-satisfaction policy, as shown in Corollary 4.5.

VI. Conclusions

In this paper we have proposed a model-free reinforcement learning (RL) algorithm to synthesize control policies that maximize the probability of satisfying high-level control objectives captured by LTL formulas. The interaction of the agent with the environment has been captured by an unknown probabilistically-labeled Markov Decision Process (MDP). We have shown that the proposed RL algorithm produces a policy that maximizes the satisfaction probability. We have also shown that even if the assigned specification cannot be satisfied, the proposed algorithm synthesizes the best possible policy. We have provided evidence via numerical experiments on the efficiency of the proposed method.

REFERENCES

[1] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *CDC and ECC*, December 2005, pp. 4885–4890.

[2] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[3] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *ICRA*, 2010, pp. 2689–2696.

[4] Y. Kantaros and M. M. Zavlanos, "Sampling-based optimal control synthesis for multi-robot systems under global temporal tasks," *IEEE Transactions on Automatic Control*, 2018. [Online]. Available: DOI:10.1109/TAC.2018.2853558

[5] ——, "Distributed intermittent connectivity control of mobile robot networks," *IEEE Transactions on Automatic Control*, vol. 62, no. 7, pp. 3109–3121, 2017.

[6] X. C. Ding, S. L. Smith, C. Belta, and D. Rus, "MDP optimal control under temporal logic constraints," in *CDC and ECC*, 2011, pp. 532–538.

[7] E. M. Wolff, U. Topcu, and R. M. Murray, "Robust control of uncertain Markov decision processes with temporal logic specifications," in *CDC*, 2012, pp. 3372–3379.

[8] X. Ding, S. L. Smith, C. Belta, and D. Rus, "Optimal control of Markov decision processes with linear temporal logic constraints," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1244–1257, 2014.

[9] M. Guo and M. M. Zavlanos, "Probabilistic motion planning under temporal tasks and soft constraints," *IEEE Transactions on Automatic Control*, 2018.

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

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

[12] E. M. Clarke, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model Checking*, 2nd ed. MIT Press, 2018.

[13] S. Sickert, J. Esparza, S. Jaax, and J. Křetínskỳ, "Limit-deterministic Büchi automata for linear temporal logic," in *CAV*. Springer, 2016, pp. 312–332.

[14] R. Alur and S. La Torre, "Deterministic generators and games for LTL fragments," *TOCL*, vol. 5, no. 1, pp. 1–25, 2004.

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

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

[17] 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*. Springer, 2014, pp. 98–114.

[18] 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*. IEEE, 2014, pp. 1091–1096.

[19] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1.

[20] J. Wang, X. Ding, M. Lahijanian, I. C. Paschalidis, and C. A. Belta, "Temporal logic motion control using actor–critic methods," *The International Journal of Robotics Research*, vol. 34, no. 10, pp. 1329–1344, 2015.

[21] Q. Gao, D. Hajinezhad, Y. Zhang, Y. Kantaros, and M. M. Zavlanos, "Reduced variance deep reinforcement learning with temporal logic specifications," 2019 (to appear).

[22] N. Fulton and A. Platzer, "Verifiably safe off-model reinforcement learning," *arXiv preprint arXiv:1902.05632*, 2019.

[23] N. Fulton, "Verifiably safe autonomy for cyber-physical systems," Ph.D. dissertation, Carnegie Mellon University Pittsburgh, PA, 2018.

[24] N. Fulton and A. Platzer, "Safe reinforcement learning via formal methods: Toward safe control through proof and learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[25] A. Platzer, "Differential dynamic logic for hybrid systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.

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

[27] ——, "Logically-constrained neural fitted Q-iteration," in *AAMAS*, 2019, pp. 2012–2014.

[28] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak, "Omega-regular objectives in model-free reinforcement learning," *arXiv preprint arXiv:1810.00950*, 2018.

[29] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science*. IEEE, 1977, pp. 46–57.

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

[31] A. Abate, M. Prandini, J. Lygeros, and S. Sastry, "Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems," *Automatica*, vol. 44, no. 11, pp. 2724–2734, 2008.

[32] A. Abate, J.-P. Katoen, J. Lygeros, and M. Prandini, "Approximate model checking of stochastic hybrid systems," *European Journal of Control*, vol. 16, no. 6, pp. 624–641, 2010.

[33] https://www.cs.ox.ac.uk/conferences/LCRL/complementarymaterials/Pacman.

[34] J. Fu and U. Topcu, "Probably approximately correct MDP learning and control with temporal logic constraints," *arXiv preprint arXiv:1404.7073*, 2014.

[35] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *CAV*. Springer, 2011, pp. 585–591.

[36] R. Durrett, *Essentials of stochastic processes*. Springer, 1999, vol. 1.

[37] V. Forejt, M. Kwiatkowska, and D. Parker, "Pareto curves for probabilistic model checking," in *ATVA*. Springer, 2012, pp. 317–332.

[38] E. A. Feinberg and J. Fei, "An inequality for variances of the discounted rewards," *Journal of Applied Probability*, vol. 46, no. 4, pp. 1209–1212, 2009.

APPENDIX

Definition 1.1: Given an LTL property ϕ and a set $\mathcal{G}$ of G-subformulas, i.e., formulas in the form $\Box(\cdot)$, we define $\phi[\mathcal{G}]$ to be the resulting formula when we substitute $true$ for every G-subformula in $\mathcal{G}$ and $\neg true$ for other G-subformulas of ϕ.

A. Proof of Proposition 4.1

Let $\mathcal{G} = \{\Box\zeta_1, ..., \Box\zeta_f\}$ be the set of all G-subformulas of ϕ. Since elements of $\mathcal{G}$ are subformulas of ϕ we can assume an ordering over $\mathcal{G}$ so that if $\Box\zeta_i$ is a subformula of $\Box\zeta_j$ then $j > i$. The accepting component of LDBA $\mathcal{Q}_D$ is a product of f DBAs $\{\mathfrak{D}_1, ..., \mathfrak{D}_f\}$ (called G-monitors) such that each $\mathfrak{D}_i = (\mathcal{Q}_i, q_{i0}, \Sigma, F_i, \delta_i)$ 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 $\delta_i : \mathcal{Q}_i \times \Sigma \to \mathcal{Q}_i$ [13]. Note that $\zeta_i[\mathcal{G}]$ has no G-subformulas. The states of the G-monitor $\mathfrak{D}_i$ are pairs of formulas where at each state, the first checks if the run satisfies $\Box\zeta_i[\mathcal{G}]$, while the second puts the next G-subformula in the ordering of $\mathcal{G}$ on hold. However, all the previous G-subformulas have been checked already and is replaced by $true$ in $\Box\zeta_i[\mathcal{G}]$. The product of the G-monitors is a deterministic generalized Büchi automaton: $\mathfrak{A}_D = (\mathcal{Q}_D, q_{D0}, \Sigma, \mathcal{F}, \delta)$, where $\mathcal{Q}_D = \mathcal{Q}_1 \times ... \times \mathcal{Q}_f$, $\Sigma = 2^{\mathcal{AP}}$, $\mathcal{F} = \{\mathcal{F}_1, ..., \mathcal{F}_f\}$, and $\delta = \delta_1 \times ... \times \delta_f$. As shown in [13], while a word w is being read by the accepting component of the LDBA, the set of G-subformulas that hold "monotonically" expands. If $w \in \{\sigma \in (2^{\mathcal{AP}})^\omega | \sigma \models \phi\}$, then eventually all G-subformulas become true.

Assume that the current state of the automaton is $q_D = (q_1, ..., q_i, ..., q_f)$ and the automaton is checking whether $\Box\zeta_i[\mathcal{G}]$ is satisfied or not, assuming that $\Box\zeta_{i-1}$ is already $true$, while putting $\Box\zeta_{i+1}[\mathcal{G}]$ on hold. At this point, the accepting frontier set is $\mathbb{A} = \{\mathcal{F}_i, \mathcal{F}_{i+1}, ..., \mathcal{F}_f\}$. Also assume the automaton returns to q_D but $\mathbb{A} \neq \{\mathcal{F}_i, \mathcal{F}_{i+1}, ..., \mathcal{F}_f\}$ then at least one accepting set $\mathcal{F}_j,\ j > i$ has been removed from $\mathbb{A}$ (Note that an accepting set $\mathcal{F}_k,\ k < i$ cannot be added since the set of satisfied G-subformulas monotonically expands). This essentially means that $\Box\zeta_j[\mathcal{G}]$ is already checked while $\Box\zeta_i[\mathcal{G}]$ is not checked yet, making $\Box\zeta_j$ a subformula of $\Box\zeta_i$. This violates the ordering of $\mathcal{G}$ and hence the assumption of $\mathbb{A}$ being time-variant is not correct.

B. Proof of Theorem 4.2

We prove this result by contradiction. Consider any policy $\overline{\boldsymbol{\mu}}$ whose traces satisfy ϕ with positive probability. Policy $\overline{\boldsymbol{\mu}}$ induces a Markov chain $\mathfrak{P}^{\overline{\boldsymbol{\mu}}}$ when it is applied over the MDP $\mathfrak{P}$. This Markov chain comprises a disjoint union between a set of transient states $\mathcal{T}_{\overline{\boldsymbol{\mu}}}$ and h sets of irreducible recurrent classes $\mathcal{R}^i_{\overline{\boldsymbol{\mu}}},\ i = 1, ..., h$ [36], namely: $\mathfrak{P}^{\overline{\boldsymbol{\mu}}} = \mathcal{T}_{\overline{\boldsymbol{\mu}}} \sqcup \mathcal{R}^1_{\overline{\boldsymbol{\mu}}} \sqcup ... \sqcup \mathcal{R}^h_{\overline{\boldsymbol{\mu}}}$. From the accepting condition of the LDBA, traces of policy $\overline{\boldsymbol{\mu}}$ satisfy ϕ with positive probability if and only if

$$\exists \mathcal{R}^a_{\overline{\boldsymbol{\mu}}} \text{ s.t. } \forall j \in \{1, ..., f\},\ \mathcal{F}^{\mathfrak{P}}_j \cap \mathcal{R}^a_{\overline{\boldsymbol{\mu}}} \neq \emptyset.$$

The recurrent class $\mathcal{R}^a_{\overline{\boldsymbol{\mu}}}$ is called an accepting recurrent class. Note that if all h recurrent classes are accepting then traces of policy $\overline{\boldsymbol{\mu}}$ satisfy ϕ with probability one. By construction of the reward function (5) the agent receives a positive reward r ever after it has reached an accepting recurrent class as it keeps visiting all the accepting sets $\mathcal{F}_j$ infinitely often.

There are two other possibilities concerning the remaining recurrent classes that are not accepting. A non-accepting recurrent class, name it $\mathcal{R}^n_{\overline{\boldsymbol{\mu}}}$, either (i) has no intersection with any accepting set $\mathcal{F}^{\mathfrak{P}}_j$, or (ii) or has intersection with some of the accepting sets but not all of them. In case (i), the agent does not visit any accepting set in the recurrent class and the likelihood of visiting accepting sets within the transient states $\mathcal{T}_{\overline{\boldsymbol{\mu}}}$ is zero since $\mathcal{Q}_D$ is invariant. In case (ii), the agent is able to visit some accepting sets but not all of them. This means that there exist always at least one accepting set $\mathcal{F}^{\mathfrak{P}}_j$ that has no intersection with $\mathcal{R}^n_{\overline{\boldsymbol{\mu}}}$ and after a finite number of times, no positive reward can be obtained, and the re-initialization of $\mathbb{A}$ in Definition 3.5 will never happen. By (7), in both cases, for any arbitrary $r > 0$, there always exists a γ such that the expected reward of a trace reaching an accepting recurrent class such as $\mathcal{R}^a_{\overline{\boldsymbol{\mu}}}$ with infinite number of positive rewards, is higher than the expected reward of any other trace.

Next, assume that the traces of optimal policy $\boldsymbol{\mu}^*$, defined in (6), do not satisfy the property ϕ. In other words, $\forall \mathcal{R}^i_{\boldsymbol{\mu}^*},\ \exists j \in \{1, ..., f\},\ \mathcal{F}^{\mathfrak{P}}_j \cap \mathcal{R}^i_{\boldsymbol{\mu}^*} = \emptyset$ and all of the recurrent classes are non-accepting. As discussed in cases (i) and (ii) above, the accepting policy $\overline{\boldsymbol{\mu}}$ has a higher expected reward than the optimal policy $\boldsymbol{\mu}^*$ due to expected infinite number of positive rewards in policy $\overline{\boldsymbol{\mu}}$. However, this contradicts the optimality of $\boldsymbol{\mu}^*$ in (6), completing the proof.

C. Proof of Theorem 4.3

We first review how the satisfaction probability is calculated traditionally when the MDP is fully known and then we show that the proposed algorithm convergence is the same. Normally when the MDP graph and transition probabilities are known, the probability of property satisfaction is often calculated via DP-based methods such as standard value iteration over the product MDP $\mathfrak{P}$ [11]. This allows to convert the satisfaction problem into a reachability problem. The goal in this reachability problem is to find the maximum (or minimum) probability of reaching *AMECs*.

The value function $V : \mathcal{S} \to [0, 1]$ in value iteration is then initialized to 0 for non-accepting maximum end components and to 1 for the rest of the MDP. Once value iteration converges then at any given state $s \in \mathcal{S}$ the optimal policy $\boldsymbol{\mu}^* : \mathcal{S} \to \mathcal{A}_{\mathfrak{P}}$ is produced by $\boldsymbol{\mu}^*(s) = \underset{a}{\text{argmax}} \sum_{s' \in \mathcal{S}} P(s, a, s')V^*(s')$, where V^* is the converged value function, representing the maximum probability of satisfying the property at state s, i.e. $U^{\boldsymbol{\mu}^*}(s)$ in our setup.

The key to compare standard model-checking methods to our method is reduction of value iteration to *basic form*. More specifically, quantitative model-checking over an MDP with a reachability predicate can be converted to a model-checking problem with an equivalent reward predicate which is called the basic form. This reduction is done by adding a one-off (or sometimes called terminal) reward of 1 upon

reaching AMECs [37]. Once this reduction is done, Bellman operation is applied over the value function (which represents the satisfaction probability) and policy $\boldsymbol{\mu}^*$ maximizes the probability of satisfying the property.

In the proposed method, when an AMEC is reached, all of the automaton accepting sets have surely been visited by policy $\boldsymbol{\mu}^*$ and an infinite number of positive rewards $r > 0$ will be given to the agent as shown in Theorem 4.2.

There are two natural ways to define the total discounted rewards [38]: (i) to interpret discounting as the coefficient in front of the reward; and (ii) to define the total discounted rewards as a terminal reward after which no reward is given and treat the update rule as if it is undiscounted. It is well-known that the expected total discounted rewards corresponding to these methods are the same; see, e.g., [38]. Therefore, without loss of generality, given any discount factor γ, and any positive reward component r, the expected discounted reward for the discounted case (the proposed algorithm) is c times the undiscounted case (value iteration) where c is a positive constant. This concludes that maximizing one is equivalent to maximizing the other.

D. Proof of Corollary 4.5

Assume that there exists no policy in $\mathfrak{M}$ whose traces can satisfy the property ϕ. Construct the induced Markov chain $\mathfrak{P}^{\boldsymbol{\mu}}$ for any arbitrary policy $\boldsymbol{\mu}$ and its associated set of transient states $\mathcal{T}_{\boldsymbol{\mu}}$ and its h sets of irreducible recurrent classes $\mathcal{R}^i_{\boldsymbol{\mu}}$: $\mathfrak{P}^{\boldsymbol{\mu}} = \mathcal{T}_{\boldsymbol{\mu}} \sqcup \mathcal{R}^1_{\boldsymbol{\mu}} \sqcup \ldots \sqcup \mathcal{R}^h_{\boldsymbol{\mu}}$. By assumption, policy $\boldsymbol{\mu}$ cannot satisfy the property and thus $\forall \mathcal{R}^i_{\boldsymbol{\mu}},\ \exists j \in \{1, ..., f\},\ \mathcal{F}^{\mathfrak{P}}_j \cap \mathcal{R}^i_{\boldsymbol{\mu}} = \emptyset$. Following the same logic as in the proof of Theorem 4.2, after a limited number of times no positive reward is given to the agent. However, by the convergence guarantees of QL, Algorithm 1 will generate a policy with the highest expected accumulated reward. By construction of the reward function in (5), this policy has the highest number of intersections with accepting sets.

E. Counter-example

We would like to emphasise that in this work and [26] $0 \leq \gamma \leq 1$ due to the fact the algorithm that we proposed is "episodic" and thus, covers the un-discounted case as well. This has been unfortunately overlooked in [28]. In the following we examine the general cases of discounted and un-discounted learning and we show that our algorithm, which is episodic, is able to output the correct action for the example provided in [28] (Fig. 4). For the sake of generality, we have parameterised the probabilities associated with action *right* and *left* with $1 - \nu$ and ν, respectively.

Recall that for a policy $\boldsymbol{\mu} : \mathcal{S} \to \mathcal{A}$ on an MDP $\mathfrak{M}$, and given a reward function R, the expected discounted reward at state s by taking action a is defined as [19]:

$$U^{\boldsymbol{\mu}}(s, a) = \mathbb{E}^{\boldsymbol{\mu}}[\sum_{n=0}^{\infty} \gamma^n \ R(s_n, \boldsymbol{\mu}(s_n))|s_0 = s, a_0 = a], \tag{10}$$

where $\mathbb{E}^{\boldsymbol{\mu}}[\cdot]$ denotes the expected value by following policy $\boldsymbol{\mu}$, and $s_1, a_1, ..., s_n, a_n$ is the sequence of state-action pairs generated by policy Pol up to time step n.

We would like to show that for some $\gamma \in [0, 1]$, $U^{\boldsymbol{\mu}}(s_0, \mathit{left}) > U^{\boldsymbol{\mu}}(s_0, \mathit{right})$. From (10), at state s_0, the expected return for each action is:

$$\begin{aligned} U^{\boldsymbol{\mu}}(s_0, \mathit{right}) &= (1 - \nu)[r + \gamma r + \gamma^2 r + ...] \\ U^{\boldsymbol{\mu}}(s_0, \mathit{left}) &= \gamma^2 r + \gamma^5 r + \gamma^8 r + ... \end{aligned} \tag{11}$$

Notice that $\boldsymbol{\mu}$ has no effect on the expected return after the agent chose to go right or left as there is only one action available in subsequent states. Let us first consider $U^{\boldsymbol{\mu}}(s_0, \mathit{right})$. The RHS is a geometric series with the initial term $(1 - \nu)r$ and ratio of γ. Thus,

$$U^{\boldsymbol{\mu}}(s_0, \mathit{right}) = (1 - \nu) r \frac{1 - \gamma^n}{1 - \gamma}. \tag{12}$$

The expected return $U^{\boldsymbol{\mu}}(s_0, \mathit{left})$ is also a geometric series such that:

$$U^{\boldsymbol{\mu}}(s_0, \mathit{left}) = \gamma^2 r \frac{1 - \gamma^{3n}}{1 - \gamma^3}. \tag{13}$$

Consider two cases (1) $0 \leq \gamma < 1$, and (2) $\gamma = 1$.

In the first case $0 \leq \gamma < 1$, as $n \to \infty$, $\gamma^n \to 0$ and $\gamma^{3n} \to 0$ and therefore, the following inequality can be solved for γ:

$$\frac{\gamma^2 r}{1 - \gamma^3} > \frac{(1 - \nu) r}{1 - \gamma} \ \longrightarrow \ \frac{\gamma^2}{1 + \gamma + \gamma^2} > 1 - \nu \ \longrightarrow$$

$$\begin{cases} \gamma < \dfrac{(-\sqrt{1/\nu^2 + 2/\nu - 3} - 1)\nu + 1}{2\nu}, \\ \\ \mathit{or} \\ \\ \gamma > \dfrac{(\sqrt{1/\nu^2 + 2/\nu - 3} - 1)\nu + 1}{2\nu}. \end{cases} \tag{14}$$

Thus, for some $\nu \in [0, 1]$, the discounted case $0 \leq \gamma < 1$ is sufficient if $\gamma > (\sqrt{1/\nu^2 + 2/\nu - 3} - 1)\nu + 1/2\nu$. However, it is possible that for some $\nu \in [0, 1]$ both conditions push γ to be outside of its range of $0 \leq \gamma < 1$ in the first case. Therefore, in the learning algorithm γ needs to be equal to 1, which brings us to the second case, that is allowed in our work thanks to the episodic nature of our algorithm.

Note that when $\gamma = 1$ we cannot derive (14) since $\lim_{n\to\infty} \gamma^n = \lim_{n\to\infty} \gamma^{3n} = 1$, and also $1 - \gamma = 0$ cannot be cancelled from both sides of the inequality. Further to this, (12) and (13) become undefined when $\gamma = 1$. From (11) though, we know with $\gamma = 1$, the summations go to infinity as $n \to \infty$. The question is, can we show that $U^{\boldsymbol{\mu}}(s_0, \mathit{left}) > U^{\boldsymbol{\mu}}(s_0, \mathit{right})$.

Recall that the convergence of QL is asymptotic and if we can show that $U^{\boldsymbol{\mu}}(s_0, \mathit{left}) > U^{\boldsymbol{\mu}}(s_0, \mathit{right})$ after a number of episodes, then essentially our algorithm can output the correct result and will choose action *left* once QL has converged.

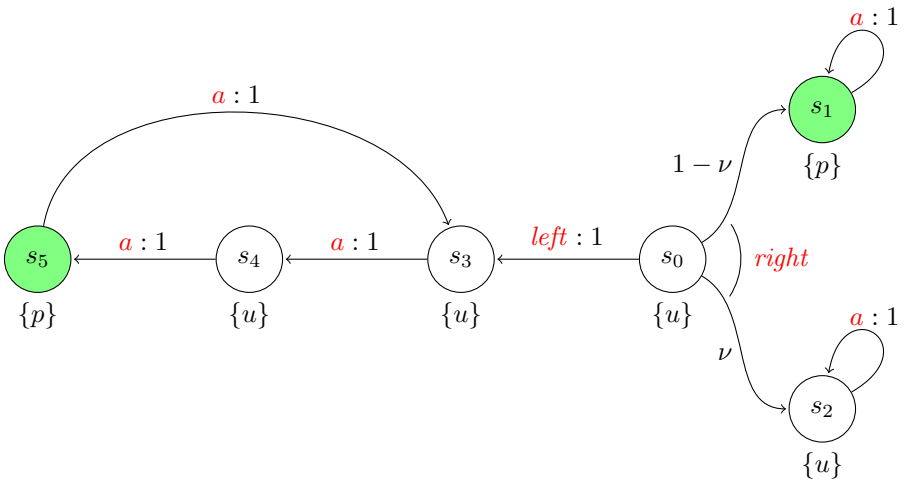


Fig. 4: Example Product MDP with $\mathcal{AP} = \{p, u\}$ with $\phi = \square\lozenge p$

To prove this claim let us consider the following limit as we push γ towards 1:

$$\lim_{\gamma\to 1}\frac{U^{\boldsymbol{\mu}}(s_0, left)}{U^{\boldsymbol{\mu}}(s_0, right)} = \lim_{\gamma\to 1}\frac{\gamma^2 r\dfrac{1-\gamma^{3n}}{1-\gamma^3}}{(1-\nu)r\dfrac{1-\gamma^n}{1-\gamma}} = \lim_{\gamma\to 1}\frac{\gamma^2 \cancel{r}\dfrac{\cancel{(1-\gamma^n)}(1+\gamma^n+\gamma^{2n})}{\cancel{(1-\gamma)}(1+\gamma+\gamma^2)}}{(1-\nu)\cancel{r}\dfrac{\cancel{1-\gamma^n}}{\cancel{1-\gamma}}} = \frac{\gamma^2}{1-\nu} \tag{15}$$

In case when $\nu = 0,\ 1-\nu = 1$ then the limit is 1, namely the algorithm is indifferent between choosing *left* or *right*. This matches with the MDP as well since going to either direction does not change the optimality of the action when $1-\nu = 1$. However, if $0 < \nu \leq 1$ then the limit is always greater than one, meaning that the expected return for taking left is greater than taking right after some finite number of episodes.