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*— We present a model-free reinforcement learning 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 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

Control synthesis for Markov Decision Processes (MDPs) under Linear Temporal Logic (LTL) specifications has been studied recently in [1]–[4]. 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 [5], [6].

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) [7]. 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 [8]. Furthermore, the semantics of the acceptance condition of a Büchi automaton is simpler than that of a Rabin automaton [4], [9], 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 [10], [11]. Specifically, [10] 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 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 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 [12]: 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 [12] is limited to policies whose traces satisfy the property with probability one. Moreover, [10]–[12] require to learn all transition probabilities of the MDP. As a result, they need a significant amount of memory to store the learned model [13]. This specific issue is addressed in [14], 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 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 [15]. 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.

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.

In [16] 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. [17] further employs DDL [18], 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 [19], [20]. Specifically, [20] propose a hybrid neural network architecture combined with LDBAs to handle MDPs with continuous state spaces. The work in [19] has been taken up more recently by [21], 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. As we have shown in the extended version of this work in [22], [21] has overlooked that the algorithm in [19] is episodic, and allows the discount factor to be equal to one. Unlike [19]–[21], 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 [5], [6].

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 [3]): 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 as $\mathcal{A}_T = a_0 a_1 \dots a_{T-1}$, 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* $\Box$, have standard syntax and meaning [23]. 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.

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 [5], [24]. We then define the probability of $\mathfrak{M}$ satisfying ϕ under policy $\boldsymbol{\xi}$ as [5], [6]

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

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

[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., [14] cannot be applied.

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. 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 [8]. By contrast, in this paper we propose to express the given LTL property as an LDBA, which results in a much more succinct automaton [7], [9]. 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 [5]): 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 [7]): 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) it 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 $P_\mathfrak{P}([x, \ell, q], a, [x', \ell', q']) = P_C(x, u, x')P_L(x', \ell')$, where $[x, \ell, q] \in \mathcal{S}$, $[x', \ell', q'] \in \mathcal{S}$, $a \in \mathcal{A}(x)$ and $q' = \delta(q, \ell')$; (v) $\mathcal{F}_\mathfrak{P} = \{(\mathcal{F}_j^\mathfrak{P}), j = 1, \ldots, f\}$ is the set of accepting states, where $\mathcal{F}_j^\mathfrak{P} = \mathcal{X} \times 2^{\mathcal{AP}} \times \mathcal{F}_j$. 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 [9]. 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

$$P_\mathfrak{P}(s, a, s') = \begin{cases} 1, & \text{if } (x = x') \wedge (\ell = \ell') \wedge (\delta(q, \varepsilon_{q'}) = q') \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

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

Given any policy $\boldsymbol{\mu}$ for $\mathfrak{P}$, we define an infinite run $\rho_\mathfrak{P}^{\boldsymbol{\mu}}$ of $\mathfrak{P}$ to be an infinite sequence of states of $\mathfrak{P}$, i.e., $\rho_\mathfrak{P}^{\boldsymbol{\mu}} = s_0 s_1 s_2 \ldots$, 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 $\rho_\mathfrak{P}^{\boldsymbol{\mu}}$ is accepting, i.e., $\boldsymbol{\mu}$ satisfies ϕ with a non-zero probability (denoted by $\boldsymbol{\mu} \models \phi$), if $\texttt{Inf}(\rho_\mathfrak{P}^{\boldsymbol{\mu}}) \cap \mathcal{F}_j^\mathfrak{P} \neq \emptyset$, $\forall j \in \{1, \ldots, f\}$.

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 $\mathcal{F}_j^\mathfrak{P}$ will be visited infinitely often.

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 without introducing extra monitoring states in the automaton; see also Section IV.

To formally present the proposed reward shaping method, we need first to introduce the *accepting frontier set* $\mathbb{A}$, which is initialized as the family set $\{F_k\}_{k=1}^f$. 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 $AF : \mathcal{Q} \times 2^{\mathcal{Q}} \to 2^{\mathcal{Q}}$ 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 $\mathcal{F}_j$ (first case). However, if $\mathbb{A} = \mathcal{F}_j$, then the output is the family set of all accepting sets of $\mathfrak{A}$ minus $\mathcal{F}_j$ (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

$$R(s, a) = \begin{cases} r & \text{if } q' \in \mathbb{A},\ s' = (x', \ell', q'), \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

In (3), 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 [7] (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.,

$$\boldsymbol{\mu}^*(s) = \arg\max_{\boldsymbol{\mu} \in \mathcal{D}} \ U^{\boldsymbol{\mu}}(s), \tag{4}$$

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

$$U^{\boldsymbol{\mu}}(s) = \mathbb{E}^{\boldsymbol{\mu}}[\sum_{n=0}^{\infty} \gamma^n \ R(s_n, \boldsymbol{\mu}(s_n)) | s_0 = s], \tag{5}$$

where $\mathbb{E}^{\boldsymbol{\mu}}[\cdot]$ denotes the expected value given that the product MDP follows the policy $\boldsymbol{\mu}$ [24], $0 \leq \gamma \leq 1$ 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 ([24]): In any finite-state MDP, such as $\mathfrak{P}$, if there exists an optimal policy, then that policy is stationary and deterministic. $\square$

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-number $:= 0$, *iteration-number* $:= 0$
while Q *is not converged* **do**
  episode-number $++$
  $s_{\text{cur}} = s_0$
  $\epsilon = 1/(episode\text{-}number)$
  while $(q \notin \mathcal{Q}_{sinks}) \wedge (iteration\text{-}number < \tau)$ **do**
    iteration-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

In order to construct $\boldsymbol{\mu}^*$, we employ episodic Q-learning (QL), a model-free RL scheme described in Algorithm 1.[2] Algorithm 1 requires as inputs (i) the LDBA $\mathfrak{A}$, (ii) the reward function R defined in (3), and (iii) the hyper-parameters of the learning algorithm.

Observe that we use an action-value function $Q : \mathcal{S} \times \mathcal{A}_{\mathfrak{P}} \to \mathbb{R}$ in Algorithm 1 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 $C : \mathcal{S} \times \mathcal{A}_{\mathfrak{P}} \to \mathbb{N}$ 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 be computed via Dynamic Programming (DP) if the MDP was fully known [13], [25], [26]. Projection of $\boldsymbol{\mu}^*$ onto the state-space of the PL-MDP yields the finite-memory policy $\boldsymbol{\xi}^*$ that solves Problem 1.

[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 [13].

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 policy if the property cannot be satisfied. The proofs of the following results are omitted owing to space limitations and can be found in [22]. 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 $\mathfrak{A}$. $\square$

As stated earlier, since QL is shown to converge to the optimal Q-function [13], it can synthesize an optimal policy with respect to the given reward function. The following theorem shows that the optimal policy produced by Algorithm 1 satisfies the given LTL property with non-zero probability.

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 (4) 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 (4), maximizes the probability of satisfying ϕ, defined in (1), as well. $\square$

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*.

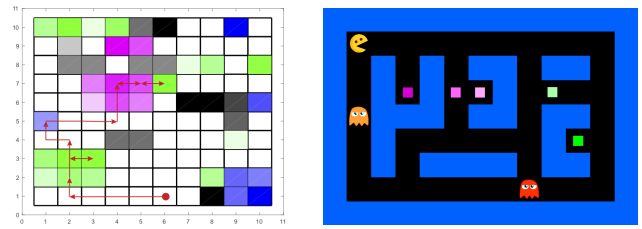
Definition 4.4 (Closeness to Satisfaction): Assume that two policies $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ do not satisfy the property ϕ. Consequently, there are accepting sets in the automaton that have no intersection with runs of the induced Markov chains $\mathfrak{P}^{\boldsymbol{\mu}_1}$ 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 $\mathfrak{P}^{\boldsymbol{\mu}_2}$. $\square$

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. $\square$

V. Experiments

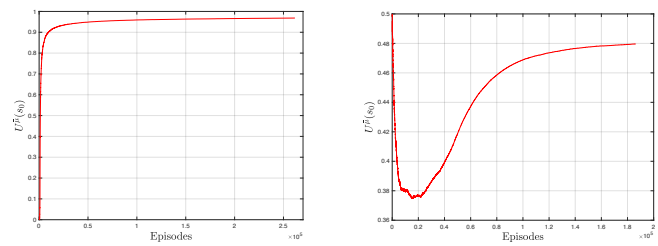
In this Section we present three case studies that illustrate the efficiency of the proposed algorithm. In the first two experiments, we consider a 10×10 discrete grid world; see Fig. 1(a). The third case study is an adaptation of the Pacman game, which is initialized in a configuration that is quite hard for the agent to solve; see Fig. 1(b).

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. The majority of the existing algorithms fail to provide a control policy when AMECs do not exist [2], [14], [27], or yield control policies without satisfaction guarantees [12].



(a) Case Studies I and II (b) Case Study III

Fig. 1: Fig 1(a): 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 colours denote the presence with non-zero probability of an obstacle, a user, target 1 and target 2, respectively. The red trajectory represents a sample path of the robot under the optimal control strategy $\boldsymbol{\xi}^*$ for the first case study. The red dot is the initial location of the robot. Fig. 1(b): Initial condition in Pacman environment. The magenta square is labeled "food1" and the green one "food2". In both figures, a higher intensity of color indicates a higher probability that the corresponding item appears.



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

Fig. 2: 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 can be found in [22].

The LTL formula considered in the first two case studies is: $\phi_1 = \Diamond(\text{target1}) \wedge \Box\Diamond(\text{target2}) \wedge \Box\Diamond(\text{user}) \wedge (\neg\text{user} \cup \text{target2}) \wedge \Box(\neg\text{obs})$. In words, ϕ_1 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 ($\Box\Diamond$target2); (iii) visit a user infinitely often where, say, the collected pictures are uploaded (captured by $\Box\Diamond$user); (iv) avoid visiting the user until a picture of target 2 has been taken; and (v) always avoid obstacles (captured by $\Box(\neg\text{obs})$). The LTL formula ϕ_1 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(a)).

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 2(a) illustrates the evolution of function $U^{\bar{\boldsymbol{\mu}}}(s_0)$ 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 the optimal control strategy $\boldsymbol{\mu}^*$ onto $\mathcal{X}$, i.e., policy $\boldsymbol{\xi}^*$, is given in Figure 1(a) (red path). In the second case study, we assume that the robot is equipped with a noisy controller and, therefore, that 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 user and target 2 and, therefore, it will violate ϕ. The optimal policy was synthesized in approximately 2 hours.

The LTL formula specifying the task for Pacman (third case study) is: $\phi_2 = \Diamond[(\text{food1} \wedge \Diamond\text{food2}) \vee (\text{food2} \wedge \Diamond\text{food1})] \wedge \Box(\neg\text{ghost})$. 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 this experiment, there is no uncertainty in the execution of actions, namely the motion of the Pacman agent is deterministic. Figure 2(b) 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 [28], or standard DP methods can be employed (if they scale) to compute the probability of satisfying ϕ for the induced Markov chain. 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. Therefore, even if the transition probabilities of the PL-MDP are known, PRISM can not generate a policy for the second case study. However, the proposed framework can synthesize the closest-to-satisfaction policy, as shown in Corollary 4.5.

VI. Conclusions

In this paper we proposed a model-free RL algorithm to synthesize control policies that maximize the probability of satisfying high-level control objectives given as LTL formulas. Theoretical results and numerical experiments support the proposed framework.

References

[1] 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.

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

[3] M. Guo and M. M. Zavlanos, "Probabilistic motion planning under temporal tasks and soft constraints," *IEEE TAC*, 2018.

[4] 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.

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

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

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

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

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

[10] J. Fu and U. Topcu, "Probably approximately correct MDP learning and control with temporal logic constraints," in *RSS*, 2014.

[11] 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.

[12] 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.

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

[14] J. Wang, X. Ding, M. Lahijanian, I. C. Paschalidis, and C. A. Belta, "Temporal logic motion control using actor–critic methods," *IJRR*, vol. 34, no. 10, pp. 1329–1344, 2015.

[15] Q. Gao, D. Hajinezhad, Y. Zhang, Y. Kantaros, and M. M. Zavlanos, "Reduced variance deep reinforcement learning with temporal logic specifications," in *ICCPS*, 2019.

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

[17] ——, "Safe reinforcement learning via formal methods: Toward safe control through proof and learning," in *AAAI*, 2018.

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

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

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

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

[22] https://www.cs.ox.ac.uk/conferences/LCRL/compmat/Pacman.

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

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

[25] 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.

[26] 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.

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

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