Logically-Constrained Reinforcement Learning

Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening

University of Oxford

Abstract. This paper proposes a Reinforcement Learning (RL) algorithm to synthesize policies for a Markov Decision Process (MDP), such that a linear time property is satisfied. We convert the property into a Limit Deterministic Büchi Automaton (LDBA), then construct a product MDP between the automaton and the original MDP. A reward function is then assigned to the states of the product automaton, according to accepting conditions of the LDBA. With this reward function, our algorithm synthesizes a policy that satisfies the linear time property: as such, the policy synthesis procedure is "constrained" by the given specification. Additionally, we show that the RL procedure sets up an online value iteration method to calculate the maximum probability of satisfying the given property, at any given state of the MDP – a convergence proof for the procedure is provided. Finally, the performance of the algorithm is evaluated via a set of numerical examples. We observe an improvement of one order of magnitude in the number of iterations required for the synthesis compared to existing approaches.

1 Introduction

Markov Decision Processes (MDPs), an extension of Markov chains, are discrete-time stochastic control processes and are extensively used for sequential decision making. MDPs are suitable for modelling decision making in situations where outcomes of actions are not fully under the control of a decision maker. Owing to this feature, MDPs have found a wide range of applications, from automatic control and AI to economics and biology [1].

Reinforcement Learning (RL) is a class of machine-learning algorithms that is widely used to solve MDPs in a variety of applications such as robotics [2–5], traffic management [6], flight control [7] and human-level game playing [8, 9]. In an RL setup, the environment feedback to the learner (or "agent") action is initially unknown. Therefore, the goal is to learn which actions to perform in a given state so that the reward signal is maximized in the long run, i.e. to learn the optimal policy [10].

Contributions: In this paper we employ RL to synthesize a control policy for an MDP such that the generated traces satisfy a Linear Temporal Logic (LTL) property. LTL, a modal logic, is able to express rich logical time-dependent properties such as safety and liveness. An LTL property can be represented as an automaton. However, LTL-to-automaton translation sometimes results in non-deterministic automata with which probabilistic model checking is in

general not meaningful. The standard solution is to use Safra's construction to determinize the automaton, which increases the size of automata dramatically [11, 12]. Another solution is to directly convert a given LTL property into a Deterministic Rabin Automaton (DRA), since the resulting automaton will by definition not be tolled by non-determinism. Nevertheless, it is known that such conversion results, in the worst case, in automata that are doubly exponential in size of formulae [13]. Although a fully non-deterministic automaton cannot be used for probabilistic model checking, it is known that automata do not need to be fully deterministic if restricted forms of non-determinism are allowed. In this paper we propose to convert the LTL property into a Limit Deterministic Büchi Automaton (LDBA) [14]. It is shown that this construction results in an exponential-sized automaton for LTL\GU (a fragment of LTL), and results in nearly the same size as a DRA for the rest of LTL formulae. Furthermore, a Büchi automaton is simpler than a Rabin automaton in terms of its acceptance conditions, which makes the algorithm implementation much simpler [15].

Once the LDBA is generated from the LTL property, we construct a synchronous product between the MDP and the resulting LDBA and then assign a reward function based on the acceptance condition of the automaton to the product MDP. By following this reward function, RL is able to generate a policy that satisfies the given LTL property. The proposed algorithm can in principle by run completely model-free, which means that we are able to synthesize policies without knowing or approximating the transition probabilities. Note that classical Dynamic Programming (DP) is of limited utility in this context, both because of its assumption of a perfect model and also because of its great computational expense [16]. In addition to the strategy synthesis problem, we employ a value iteration method to calculate the probability of satisfaction of the LTL property. Use of RL for policy generation allows the value iteration algorithm to focus on parts of state space that are relevant to the property. This results in a faster calculation of probability values when compared to DP, where in contrast these values are globally updated over the whole state space.

Related Work: The problem of control synthesis for temporal logic is considered in a number of works. In [17], it is assumed that the MDP has unknown transition probabilities and the solution is to build a probably approximately correct MDP. The logical property is converted to a DRA and a product MDP is generated. The optimal policy is generated via a value iteration method. Comparatively, [6] employed RL to generate the optimal policy. Using DRA, a robust control policy is proposed in [18], that maximizes the worst-case probability of satisfying the specification over all transition probabilities.

In [19], the problem of minimizing the average expected cost per stage is combined with the control synthesis problem. The problem of synthesizing a policy that satisfies a temporal logic specification and at the same time optimizes a performance criterion is considered in [20]. The authors seperate the problem into two sub-problems: extracting a (maximally) permissive strategy for the agent and then quantifying the performance criterion as a reward function and computing an optimal strategy for the agent within the operating envelope allowed

by the permissive strategy. Comparatively, in [21], the authors first compute safe, permissive strategies with respect to a reachability property. Then, under these constrained strategies, reinforcement learning is applied to synthesize a policy that satisfies an expected cost property.

Truncated LTL is proposed in [22] as the specification language, and a policy-search method is used for policy generation. In [23], scLTL is proposed for mission specification, which is expressible by a deterministic finite automaton. A product MDP is then constructed and a linear programming solver is used to find the optimal policy. PCTL specifications are also investigated in [24], where a linear optimization solution is used to synthesize a control policy. In [25], an automated method is proposed to verify and repair the policies that are generated by RL with respect to a PCTL formula. The idea is to find the induced Markov chain of the policy and then feed it to a probabilistic model checker. In [26], the challenges of continuous reinforcement learning have been addressed by letting the agent plan ahead in real time using constrained optimization: in each time step, the agent makes decisions by efficiently solving a sequence of finite horizon optimization problems in real time to plan ahead.

The problem of policy generation via maximizing probability of satisfying the specification for unbounded reachability property is investigated in [27]. The policy generation in this work relies on approximate DP even when MDP transition probabilities are unknown. This requires a mechanism to approximate these probabilities, and the optimality of the generated policy highly depends on the accuracy of this approximation. Therefore, as mentioned in [27], a sufficiently large number of simulations has to be executed to make sure that the probability approximations are accurate enough. However, in our model-free RL approach, the optimal policy is obtained without any approximation mechanism, which makes our algorithm simpler to implement and leads to a faster policy generation. Finally, via LTL-to-DRA conversion, which will result in a double exponential blowup, it is mentioned in [27] that the algorithm can be extended to the problem of control synthesis for LTL specifications.

The concept of shielding is employed in [28] to synthesize a reactive system that ensures that the agent stays safe during and after learning. This approach is closely related to teacher-guided RL [29], since a shield can be considered as a teacher, which provides safe actions only if absolutely necessary. However, unlike our focus on full LTL expressivity, [28] adopted the safety fragment of LTL as the specification language. To express the specification, [28] uses DFAs and then translates the problem into a safety game. The game is played by the environment and the agent. In every state of the game, the environment chooses an input, and then the agent chooses some output. The game is won by the agent if only safe states are visited during the play. However, the generated policy always needs the shield to be online, as the shield maps every unsafe action to a safe action.

Organization: The organization of this paper is as follows: Section 2 reviews basic concepts and definitions. In Section 3, we discuss the policy synthesis problem and we propose a method to constrain it. In Section 4, we discuss an online value iteration method to calculate the maximum probability of satisfying

the LTL property at any given state. Case studies are provided in Section 5 to quantify the performance of the proposed algorithms. The appendix includes proofs of theorems and propositions we proposed in this paper.

2 Background

2.1 Problem Framework

Definition 1 (Markov Decision Process (MDP)). *An MDP* $\boldsymbol{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ *is a tuple over a finite set of states* $\mathcal{S}$ *where* $\mathcal{A}$ *is a finite set of actions,* s_0 *is the initial state and* $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is the transition probability function which determines probability of moving from a given state to another by taking an action.* $\mathcal{AP}$ *is a finite set of atomic propositions and a labelling function* $L : \mathcal{S} \to 2^{\mathcal{AP}}$ *assigns to each state* $s \in \mathcal{S}$ *a set of atomic propositions* $L(s) \subseteq 2^{\mathcal{AP}}$*. Assume that the set of available actions at state* s *is* $\mathcal{A}_s \subseteq \mathcal{A}$ *(i.e.* $\mathcal{A}_s = \{a : P(s, a, s') \neq 0,\ s' \in \mathcal{S}\}$*). We use* $s \xrightarrow{a} s'$ *to denote a transition from state* $s \in \mathcal{S}$ *to state* $s' \in \mathcal{S}$ *by action* $a \in \mathcal{A}_s$*.* ⌟

Definition 2 (Stationary Policy). *A stationary randomized policy* $Pol : \mathcal{S} \times \mathcal{A} \to [0, 1]$ *is a mapping from each state* $s \in \mathcal{S}$*, and action* $a \in \mathcal{A}$ *to the probability of taking action* a *in state* s*. A deterministic policy is a degenerate case of a randomized policy which outputs a single action at given state, that is* $\exists a \in \mathcal{A} \mid Pol(s, a) = 1$*.*

Let the MDP $\mathbf{M}$ be a model that formulates the interaction between the agent and its environment. The immediate reward received by the agent from the environment after performing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ is denoted by a function called reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We assume this reward function is initially unknown for the agent and has to be discovered. Note that the reward function for us as the designer is known in the sense that we know over which state (or under what circumstances) the agent will receive a relevant reward. The reward function specify what the agent needs to achieve, not how to achieve it. Thus, the objective is that we would like the agent itself to make an effort to come up with a solution.

Q-learning (QL), a sub-class of RL algorithms, is extensively used to solve MDPs with unknown reward functions [16]. For each state $s \in \mathcal{S}$ and for any available action $a \in \mathcal{A}_s$, QL assigns a quantitative value $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which is initialized with an arbitrary fixed value for all state-action pairs. As the agent starts learning and receiving rewards, the Q function is updated by the following rule:

$$\begin{cases} Q(s, a) \leftarrow Q(s, a) + \mu[R(s, a) + \gamma \max\limits_{a' \in \mathcal{A}_s} (Q(s', a')) - Q(s, a)], \\ Q(s'', a'') \leftarrow Q(s'', a''), \end{cases}$$

where $Q(s, a)$ is the Q-value corresponding to state-action (s, a), $\mu \in (0, 1]$ is the step size or learning rate, $R(s, a)$ is the agent's realized reward for performing

action a in state s, $\gamma \in [0,1)$ (and $[0,1]$ in episodic MDPs) is a coefficient called discount factor, s' is the state after performing action a, and (s'', a'') refers to any state-action pair other than (s, a).

Definition 3 (Expected Discounted Utility). *For a policy Pol on an MDP* $\boldsymbol{M}$*, the expected discounted utility is defined as [16]:*

$$U^{Pol}(s) = \mathbb{E}^{Pol}[\sum_{n=0}^{\infty} \gamma^n \ R(S_n, Pol(S_n))|S_0 = s],$$

where $\mathbb{E}^{Pol}[\cdot]$ *denotes the expected value given that the agent follows policy Pol,* γ *is the discount factor, and* $S_0, ..., S_n$ *is the sequence of states generated by policy Pol up to time step* n*.* ⌟

Definition 4 (Optimal Policy). *Optimal policy* Pol^* *is defined as follows:*

$$Pol^*(s) = \arg \max_{Pol \in \mathcal{D}} \ U^{Pol}(s),$$

where $\mathcal{D}$ *is the set of all stationary deterministic policies over* $\mathcal{S}$*.* ⌟

Theorem 1. *In any finite-state MDP* $\boldsymbol{M}$*, if there exists an optimal policy, then that policy is stationary and deterministic [1].* ⌟

It has been proven in [30] that Q values in QL algorithm converges to Q^* such that $U^{Pol^*}(s) = \max_{a \in \mathcal{A}_s} Q^*(s, a)$. Now suppose that the agent is in state s. The simplest method to select an optimal action at each state s (i.e. to synthesize an optimal policy) is to choose an action that yields the highest Q-value at each state (i.e. a greedy action selection). Then the optimal policy function $Pol^* : \mathcal{S} \to \mathcal{A}$ is

$$Pol^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} \ Q(s, a).$$

Regardless of the policy synthesis method, the biggest merit of QL is that the expected rewards of the state-action pairs can be compared without requiring a knowledge of the reward function.

3 Constrained Policy Synthesis

So far, we discussed how the agent can achieve the desired objective by following a reward function over the states of the MDP. The reward function can guide the agent to behave in a certain way during learning. In order to specify a set of desirable constraints (i.e. properties) over the agent evolution we employ Linear Temporal Logic (LTL). An LTL formula can express a wide range of properties, such as safety and persistence. The set of LTL formulae over the set of atomic proposition $\mathcal{AP}$ is defined as

$$\varphi ::= true \mid \alpha \in \mathcal{AP} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi \cup \varphi. \tag{1}$$

In the following we define LTL formulae interpreted over MDPs.

Definition 5 (Path). *In an MDP $\boldsymbol{M}$, an infinite path ρ starting at s_0 is a sequence of states $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ...$ such that every transition $s_i \xrightarrow{a_i} s_{i+1}$ is possible in $\boldsymbol{M}$, i.e. $P(s_i, a_i, s_{i+1}) > 0$. We might also denote ρ as $s_0..$ to emphasize that ρ starts from s_0. A finite path is a finite prefix of an infinite path.* ⌟

Given a path ρ, the i-th state of ρ is denoted by $\rho[i]$ where $\rho[i] = s_i$. Furthermore, the i-th suffix of ρ is $\rho[i..]$ where $\rho[i..] = s_i \rightarrow s_{i+1} \rightarrow s_{i+2} \rightarrow s_{i+3} \rightarrow ...$.

Definition 6 (LTL Semantics). *For an LTL formula φ and for a path ρ, the satisfaction relation $\rho \models \varphi$ is defined as*

$$\rho \models \alpha \in \mathcal{AP} \iff \alpha \in L(\rho[0]),$$
$$\rho \models \varphi_1 \wedge \varphi_2 \iff \rho \models \varphi_1 \wedge \rho \models \varphi_2,$$
$$\rho \models \neg\varphi \iff \rho \not\models \varphi,$$
$$\rho \models \bigcirc\varphi \iff \rho[1..] \models \varphi$$
$$\rho \models \varphi_1 \cup \varphi_2 \iff \exists j \in \mathbb{N} \cup \{0\} \text{ s.t. } \rho[j..] \models \varphi_2 \text{ and } \forall i,\ 0 \leq i < j,\ \rho[i..] \models \varphi_1.$$

⌟

Using the until operator we are able to define two temporal modalities: (1) eventually, $\Diamond\varphi = true \cup \varphi$; and (2) always, $\Box\varphi = \neg\Diamond\neg\varphi$.

Definition 7 (Probability of Satisfying an LTL Formula). *Starting from s_0, for a sequence $\rho = s_0 \rightarrow s_1 \rightarrow ...$ we define the probability of satisfying formula φ as*

$$Pr^{Pol}(s_0.. \models \varphi),$$

where $Pol : \mathcal{S} \rightarrow \mathcal{A}$ is a deterministic policy that at a given state s_i determines the successor state s_{i+1} by taking action a_i. ⌟

An LTL formula φ also induces a linear time property over $\mathcal{AP}$ specifying the following set of words:

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

Using an LTL formula we can specify a set of constrains (i.e. properties) over the sequence of states that are generated by the policy in the MDP. Once a policy is selected (e.g. Pol) then at each state of an MDP $\mathbf{M}$, it is clear which action has to be taken. Hence, the MDP $\mathbf{M}$ is reduced to a Markov chain, which we denote by $\mathbf{M}^{Pol}$. Through this paper, when we say that policy Pol satisfies φ, we mean that all the runs of $\mathbf{M}^{Pol}$ satisfy φ.

For an LTL formula φ, an alternative method to express the set of associated words, i.e. $Words(\varphi)$, is to employ an LDBA [14]. We need to first define a Generalized Büchi Automaton (GBA) and then we formally introduce LDBA [14].

Definition 8 (Generalized Büchi Automaton). *A GBA $\boldsymbol{N} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is a structure where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $\Sigma = 2^{\mathcal{AP}}$ is a finite alphabet, $\mathcal{F} = \{F_1, ..., F_f\}$ is the set of accepting conditions where $F_j \subset \mathcal{Q}, 1 \leq j \leq f$, and $\Delta : \mathcal{Q} \times \Sigma \rightarrow 2^{\mathcal{Q}}$ is a transition relation.* ⌟

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

$$inf(\theta) \cap F_j \neq \emptyset, \tag{2}$$

where $inf(\theta)$ is the set of states that are visited infinitely often in the sequence θ. The accepted language of the GBA $\mathbf{N}$ is the set of all infinite words accepted by the GBA $\mathbf{N}$ and it is denoted by $\mathcal{L}_\omega(\mathbf{N})$.

Definition 9 (LDBA). *A GBA $\boldsymbol{N} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is limit deterministic if $\mathcal{Q}$ can be partitioned into two disjoint sets $\mathcal{Q} = \mathcal{Q}_N \cup \mathcal{Q}_D$, such that [14]:*

- *$\Delta(q, \alpha) \subset \mathcal{Q}_D$ and $|\Delta(q, \alpha)| = 1$ for every state $q \in \mathcal{Q}_D$ and for every $\alpha \in \Sigma$,*
- *for every $F_j \in \mathcal{F}$, $F_j \subset \mathcal{Q}_D$.* ⌟

Remark 1. The LTL-to-LDBA algorithm proposed in [14] results in an automaton with two parts: initial ($\mathcal{Q}_N$) and accepting ($\mathcal{Q}_D$). Both the initial part and the accepting part are deterministic and there are non-deterministic ε-transitions between the initial part and the accepting part. An ε-transition allows an automaton to change its state spontaneously and without reading an input symbol. ⌟

Recall that the MDP models the agent-environment interaction. We are interested in synthesizing a policy for this MDP such that the resulting traces satisfy a given LTL property. We propose to generate an LDBA from the desired LTL property and then to construct a product between the MDP and the Büchi automaton.

Definition 10 (Product MDP). *Given an MDP $\boldsymbol{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ and an LDBA $\boldsymbol{N} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ $\Sigma = 2^{\mathcal{AP}}$, the product MDP is defined as $\boldsymbol{M} \otimes \boldsymbol{N} = \boldsymbol{M_N} = (\mathcal{S}^\otimes, \mathcal{A}, s_0^\otimes, P^\otimes, \mathcal{AP}^\otimes, L^\otimes)$, where $\mathcal{S}^\otimes = \mathcal{S} \times \mathcal{Q}$, $s_0^\otimes = (s_0, q_0)$, $\mathcal{AP}^\otimes = \mathcal{Q}$, $L^\otimes = \mathcal{S} \times \mathcal{Q} \to 2^{\mathcal{Q}}$ such that $L^\otimes(s, q) = q$, $P^\otimes : \mathcal{S}^\otimes \times \mathcal{A} \times \mathcal{S}^\otimes \to [0, 1]$ is the transition probability function such that $(s_i \xrightarrow{a} s_j) \wedge (q_i \xrightarrow{L(s_j)} q_j) \Rightarrow P^\otimes((s_i, q_i), a, (s_j, q_j)) = P(s_i, a, s_j)$. Over the states of the product MDP we also define accepting condition $\mathcal{F}^\otimes$ where $F_j^\otimes \in \mathcal{F}^\otimes, 1 \leq j \leq f$ is a subset of $\mathcal{S}^\otimes$ such that for each $s^\otimes = (s, q) \in F_j^\otimes,\ q \in F_j$.* ⌟

Remark 2. 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 [31]:

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

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

- The transition probabilities of ε-transitions are given by

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

⌟

Note that the resulting product is an MDP over which we can define a reward function and run the QL algorithm. Intuitively, the product MDP encompasses transition relations of the original MDP and also the structure of the Büchi automaton, and thus inherits characteristics of both. Therefore, a proper reward assignment can lead the RL agent to find a policy that is optimal and respects both the original MDP and the LTL property φ. Before introducing the reward assignment that we considered in this paper, we need to first present the ensuing notions.

Definition 11 (End Component). *An end component of MDP* $\boldsymbol{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ *is a directed graph induced by a pair* (S, A) *such that it is strongly connected [32].* ⌟

Definition 12 (Maximal End Component (MEC)). *An end component* (S, A) *of MDP* $\boldsymbol{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ *is maximal if there exists no end component* (S', A') *such that* $(S, A) \neq (S', A')$ *and* $S \subset S'$ *and* $A_s \subset A'_s$ *for all* $s \in S$ *[32].* ⌟

Definition 13 (Accepting Maximal End Component (AMEC)). *In a product MDP* $\boldsymbol{M_N} = (\mathcal{S}^{\otimes}, \mathcal{A}, s_0^{\otimes}, P^{\otimes}, \mathcal{AP}^{\otimes}, L^{\otimes}, \mathcal{F}^{\otimes})$ *where* $\boldsymbol{N}$ *is a GBA, a MEC* $(S^{\otimes}, A^{\otimes})$ *is an AMEC if for each* $F_j^{\otimes}$

$$S^{\otimes} \cap F_j^{\otimes} \neq \emptyset.$$

We also define AMECs as the set of all accepting maximal end components. ⌟

We propose the following reward assignment over the states of the product MDP:

$$R(s^{\otimes}, a) = \begin{cases} r_p & \text{if } s^{\otimes\prime} \in \text{AMECs}, \\ r_n & \text{otherwise.} \end{cases}$$

where $s^{\otimes\prime}$ is the state that is reached from state $s^{\otimes}$ by taking action a, $r_p > 0$ is a positive reward, and $r_n = 0$ is a neutral reward. Observe that since the reward for all the states in an AMEC is the same, then the optimal policy Pol^* generated from the Q-values is indifferent to select a specific action and with Pol^* there is a positive probability for each state in the AMEC to be selected.

Theorem 2. *Let MDP* $\boldsymbol{M_N}$ *be the product of an MDP* $\boldsymbol{M}$ *and an automaton* $\boldsymbol{N}$ *where* $\boldsymbol{N}$ *is the LDBA associated with the desired LTL property* φ*. Any optimal policy which optimizes the expected utility will satisfy the property.* ⌟

Remark 3. When the number of accepting sets in the LDBA is only one, i.e. $|\mathcal{F}| = 1$, there is no need to calculate MECs and find the accepting one. In this case we only assign positive rewards to the accepting set $\mathcal{F}$, and the optimal policy will satisfy the property. The idea of the proof is similar to Theorem 2. Furthermore, in this case there is no need to explicitly build the product MDP. The automaton transitions can be executed on-the-fly as the agent reads the labels of the MDP states. Thus, the agent is aware of the automaton state and receives a positive reward whenever it hits $\mathcal{F}$. ⌟

Once the optimal policy $Pol^* : \mathcal{S}^{\otimes} \rightarrow \mathcal{A}$ is obtained for the product MDP $\mathbf{M_N}$, it induces a Markov chain, which we denote by $\mathbf{M}_{\mathbf{N}}^{Pol^*}$. If we monitor the traces of states in $\mathcal{S}$, the optimal policy Pol^* also induces a Markov chain $\mathbf{M}^{Pol^*}$ over the original MDP $\mathbf{M}$. This induced Markov chain $\mathbf{M}^{Pol^*}$ satisfies the desired LTL property with probability

$$Pr^{Pol^*}(s_0^{\otimes}.. \models \varphi) = Pr(\{\rho : \ \forall j \ inf(\rho) \cap F_j^{\otimes} \neq \emptyset\}),$$

where ρ is a run of $\mathbf{M}_{\mathbf{N}}^{Pol^*}$ initialized at $s_0^{\otimes}$.

4 Probability of Property Satisfaction

Over the states of the MDP $\mathbf{M}$, define a function $PSP^* : \mathcal{S} \rightarrow [0, 1]$, which at a given state $s \in \mathcal{S}$ returns the maximum probability of satisfying the desired property:

$$PSP^*(s) = \max_{Pol \in \mathcal{D}} Pr^{Pol}(s.. \models \varphi),$$

where $Pr^{Pol}(s.. \models \varphi)$ is the probability of satisfying φ at state s if we use the policy Pol to determine subsequent states and $\mathcal{D}$ is the set of all stationary policies over $\mathcal{S}$. This probability can be calculated by model-based tools (e.g. PRISM [33]) or MEC-based methods (e.g. [34]). However, in this section we propose a local value iteration method that calculates this probability during learning.

Consider a function $PSP : \mathcal{S}^{\otimes} \rightarrow [0, 1]$. For a given state $s^{\otimes}$ the PSP function is initialized as $PSP(s^{\otimes}) = 0$ if $s^{\otimes}$ is in a non-accepting sink MEC. Otherwise it is initialized as $PSP(s^{\otimes}) = 1$. Recall that the set of available actions at state $s^{\otimes}$ is $\mathcal{A}_{s^{\otimes}} \subseteq \mathcal{A}$ (i.e. $\mathcal{A}_{s^{\otimes}} = \{a : P(s^{\otimes}, a, s^{\otimes\prime}) \neq 0, \ s^{\otimes\prime} \in \mathcal{S}^{\otimes}\}$). Once state $s^{\otimes}$ is visited, $PSP(s^{\otimes})$ is updated by

$$PSP(s^{\otimes}) \leftarrow \max_{a \in \mathcal{A}_{s^{\otimes}}} \sum_{s^{\otimes\prime} \in \mathcal{S}^{\otimes}} P(s^{\otimes}, a, s^{\otimes\prime}) PSP(s^{\otimes\prime}).$$

In the following we prove that the proposed update rule for PSP converges to the PSP^*.

Definition 14 (Bellman's operation [35]). *For any vector such as $PSP = (PSP(s_1), ..., PSP(s_{|\mathcal{S}|}))$ in the MDP $\boldsymbol{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$, the Bellman DP operation T PSP is defined as:*

$$T \ PSP(s_i) = \max_{a \in \mathcal{A}_{s_i}} \sum_{s' \in \mathcal{S}} P(s_i, a, s') PSP(s'). \tag{3}$$

⌟

Definition 15. *The optimal value vector is denoted by $PSP^* = (PSP^*(s_1), ..., PSP^*(s_{|\mathcal{S}|}))$, where $PSP^*(s_i)$ is the optimal value starting from state s_i and*

$$PSP^*(s_i) = \max_{Pol \in \mathcal{D}} PSP^{Pol}(s_i),$$

where $PSP^{Pol}(s_i)$ is the value of state s_i if we use the policy Pol to determine subsequent states. ⌟

Proposition 1. *The optimal value vector PSP^* has finite components and satisfies the following equation*

$$PSP^* = T\ PSP^*.$$

Additionally, PSP^* is the only *solution of the equation $PSP = T\ PSP$ [35].* ⌟

Definition 16 (Gauss-Seidel Asynchronous Value Iteration (AVI)). *In the standard value iteration method the value estimation is simultaneously updated for all states. However, an alternative method is to update the value for one state at a time. This method is known as asynchronous value iteration. We denote this operation by F. Assume for state s_1, the value is updated at the current time. Thus,*

$$F\ PSP(s_1) = \max_{a \in \mathcal{A}_{s_1}} \{\sum_{s' \in \mathcal{S}} P(s_1, a, s') PSP(s')\}, \tag{4}$$

and for all $s_i \neq s_1$

$$\begin{aligned} F\ PSP(s_i) = \max_{a \in \mathcal{A}_{s_i}} \{ & \sum_{s' \in \{s_1, \ldots, s_{i-1}\}} P(s_i, a, s') \\ & F\ PSP(s') + \sum_{s' \in \{s_i, \ldots, s_{|\mathcal{S}|}\}} P(s_i, a, s') PSP(s')\}. \end{aligned} \tag{5}$$

So, basically, by (5) we update the value of PSP state by state and use the calculated value for the next step [35]. ⌟

Proposition 2. *Let $k_0, k_1, \ldots$ be an increasing sequence of iteration indices such that $k_0 = 0$ and each state is updated at least once between iterations k_m and $k_{m+1} - 1$, for all $m = 0, 1, \ldots$. Then the sequence of value vectors generated by AVI converges to PSP^* [35].* ⌟

Proposition 3 (Convergence). *Under assumptions of Proposition 2, our proposed method is an AVI and will converge to PSP^*, i.e. the probability that can be calculated by model-based tools such as PRISM.* ⌟

Lemma 1. *F is a contraction mapping with respect to the infinity norm. In other words, for any two value vectors PSP and PSP' [35]:*

$$||F\ PSP - F\ PSP'||_\infty \leq ||PSP - PSP'||_\infty. \tag{6}$$

⌟

Proposition 4. *[Convergence rate] Our proposed value iteration update rule converges at least after N iterations where $N = [L/D] + 1$ and $L = ||PSP - PSP^*||_\infty$, $D = ||F\ PSP - PSP||_\infty$.* ⌟

We conclude this section by presenting the structure of our algorithm in Algorithm 1.

Algorithm 1: Logically-Constrained Reinforcement Learning (LCRL)

input : MDP $\mathbf{M}$, LTL specification, desired error ϵ, arbitrary $E > \epsilon$, initialize $Q : \mathcal{S}^{\otimes} \times \mathcal{A}^{\otimes} \to \mathbb{R}$ and $PSP : \mathcal{S}^{\otimes} \to [0, 1]$

output : PSP^* and Pol^*

Convert the desired LTL property to an LDBA $\mathbf{N}$
Build the product MDP $\mathbf{M_N}$
Initialize $Episode\text{-}Number := 0$
while $E > \epsilon$ **do**
  $Episode\text{-}Number++$
  $s^{\otimes} = (s_0, q_0)$
  $OPSP = PSP$
  repeat
    $\mathcal{A}_{s^{\otimes}} = \{a : P^{\otimes}(s^{\otimes}, a, s^{\otimes\prime}) \neq 0,\ s^{\otimes\prime} \in \mathcal{S}^{\otimes}\}$
    choose $a_* = Pol(s^{\otimes}) = \text{argmax}_{a \in \mathcal{A}_{s^{\otimes}}} Q(s^{\otimes}, a)$
    move to $s^{\otimes}_*$ by a_*
    receive the reward $R(s^{\otimes}, a_*)$
    $Q(s^{\otimes}, a_*) \leftarrow Q(s^{\otimes}, a_*) + \mu_n[R(s^{\otimes}, a_*) - Q(s^{\otimes}, a_*) + \gamma \max_{a'}(Q(s^{\otimes}_*, a'))]$
    $s^{\otimes} = s^{\otimes}_*$
    $PSP(s^{\otimes}) \leftarrow \max_{a \in \mathcal{A}_{s^{\otimes}}} \sum_{s^{\otimes\prime} \in \mathcal{S}^{\otimes}} P(s^{\otimes}, a, s^{\otimes\prime}) \times PSP(s^{\otimes\prime})$
  until *end of trial*
  $E = ||PSP - OPSP||_{\infty}$
end

5 Experimental Results

5.1 Description of the Benchmark

Let the area be an $L \times L$ square over which a two-dimensional grid is applied. The interaction between the robot and the grid (as its environment) can be modelled as an MDP. In this MDP, the robot location is a state $s \in \mathcal{S}$. At each state $s \in \mathcal{S}$ the robot has a set of actions $\mathcal{A}_s \subseteq \mathcal{A} = \{left, right, up, down, stay\}$ by which the robot is able to move to other states (e.g. s') with the probability of $P(s, a, s'), a \in \mathcal{A}_s$. In this particular example, at each state $s \in \mathcal{S}$, the robot available actions are either to move to a neighbour state $s' \in \mathcal{S}$ or to stay at the state s. In this example, if not otherwise specified, we assume for each action the robot chooses, there is a probability of 85% that the action takes the robot to the correct state and 15% that the action takes the robot to a random state in its neighbourhood (including its current state).

A labelling function $L : \mathcal{S} \to 2^{\mathcal{AP}}$ assigns to each state $s \in \mathcal{S}$ a set of atomic propositions $L(s) \subseteq \mathcal{AP}$. We assume that in each state s the robot is aware of the labels of the neighbouring states, which is a realistic assumption. We consider two 40×40 regions and one 3×3 region with different labellings as in Fig. 1. In Region 3 and in state target, the subsequent state after performing action *stay* is always the state target itself. Note that all the actions are not active in Region 3.

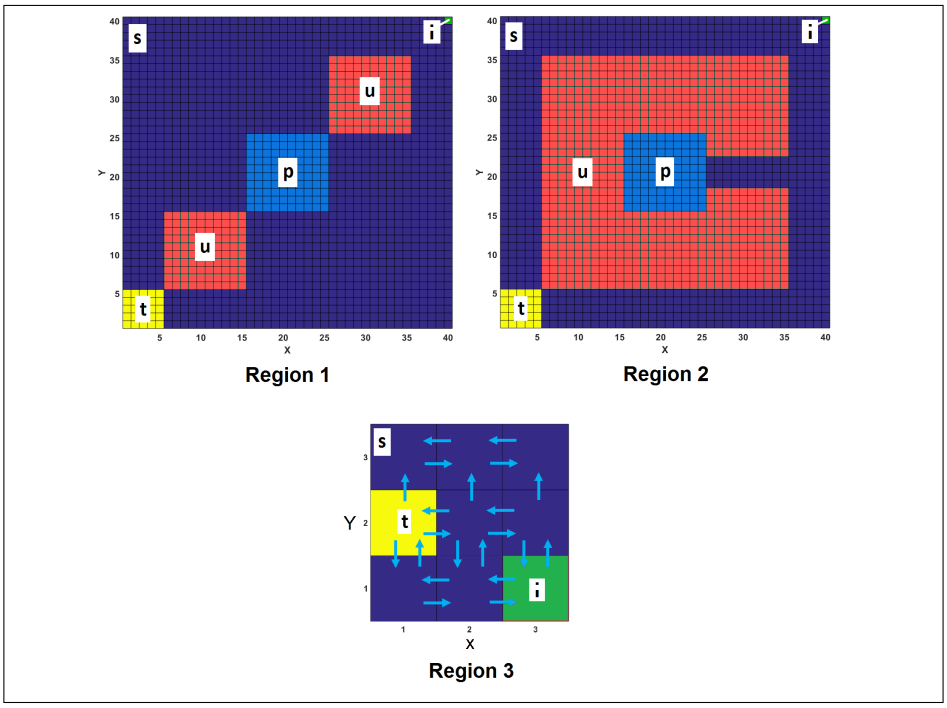

Fig. 1: MDP labelling - green: initial state (i), dark blue: safe (s), red: unsafe (u), light blue: pre-target (p), yellow: target (t)

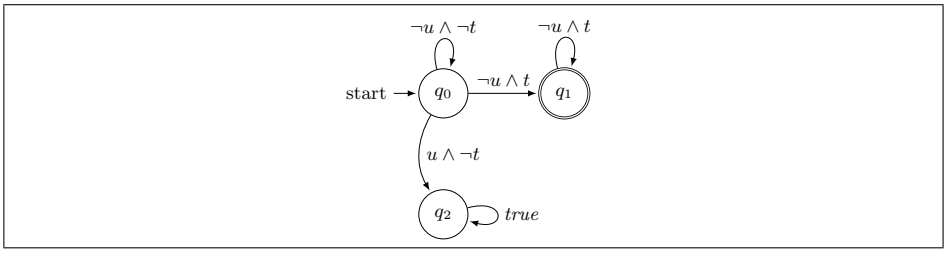

Fig. 2: LDBA for (7) with removed transitions labelled $t \wedge u$ (since it is impossible to be at target and unsafe at the same time)

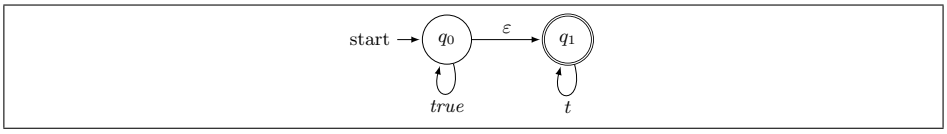

Fig. 3: LDBA for (8)

5.2 Properties

An LTL property allows us to specify a mission task in a rich time-dependent formal language. By employing LTL, we are able to express complex control

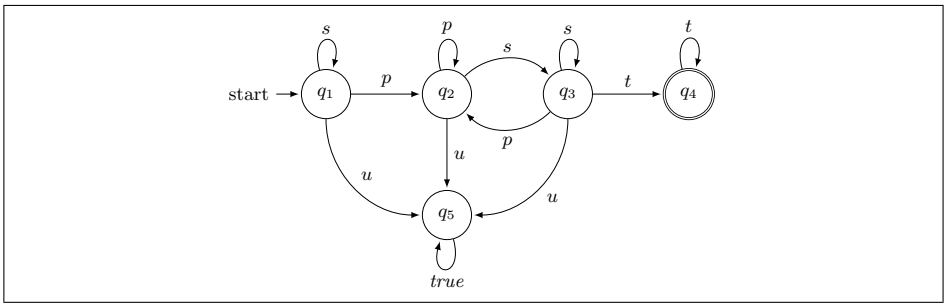

Fig. 4: LDBA for (9)

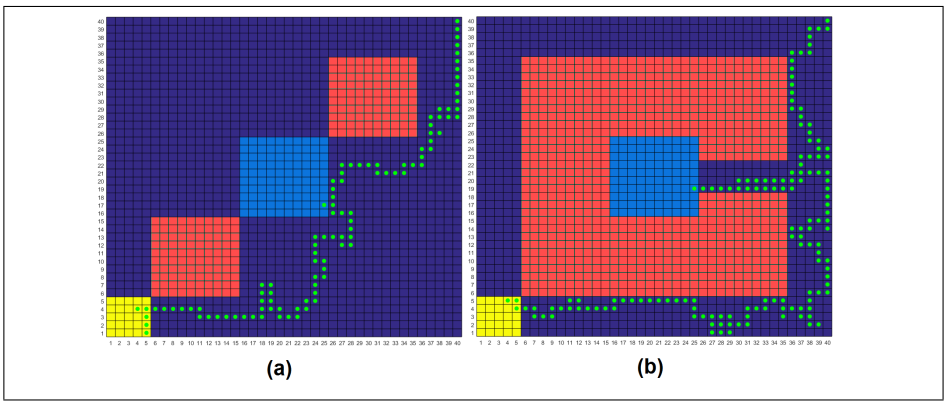

Fig. 5: Simulation results for (9) in (a) Region 1 and (b) Region 2

objectives that are hard or impossible to be expressed in rewards in RL. The first control objective in this numerical example is expressed by the following LTL formula:

$$\Diamond t \wedge \Box(u \rightarrow \Box u) \wedge \Box(t \rightarrow \Box t), \tag{7}$$

and

$$\Diamond \Box t, \tag{8}$$

where t stands for "target" and u stands for "unsafe". In this case study, we also investigate the following regular expression:

$$s^* p p^* s(s|p)^* t^\omega, \tag{9}$$

where s stands for "safe", and p refers to the area that has to be visited before visiting the area with label t, which stands for "target". We can build the Büchi automaton associated with (7), (8) and (9) as in Fig. 2, Fig. 3 and Fig. 4. Note that the LDBA in Fig. 2 has a non-deterministic ε transition.

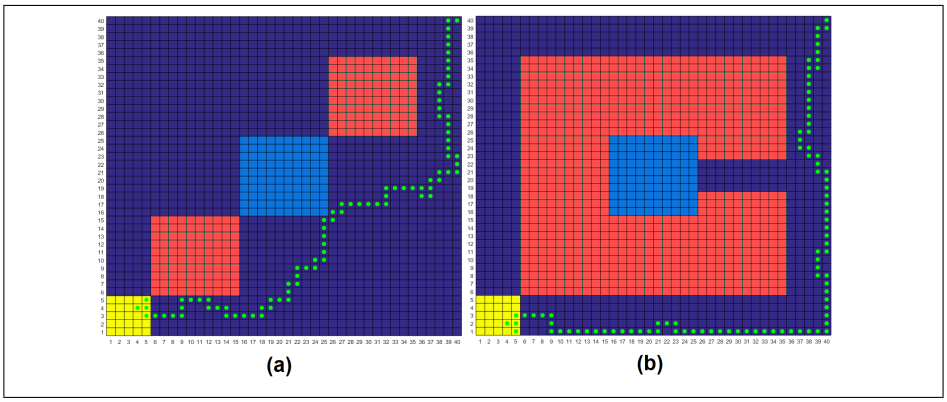

Fig. 6: Simulation results for (7) in (a) Region 1 and (b) Region 2

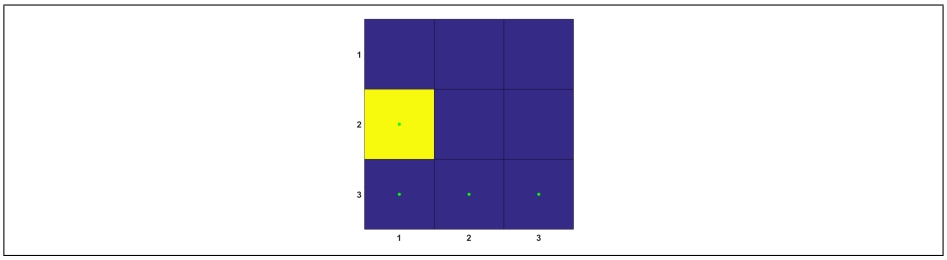

Fig. 7: Simulation results for (8) in Region 3

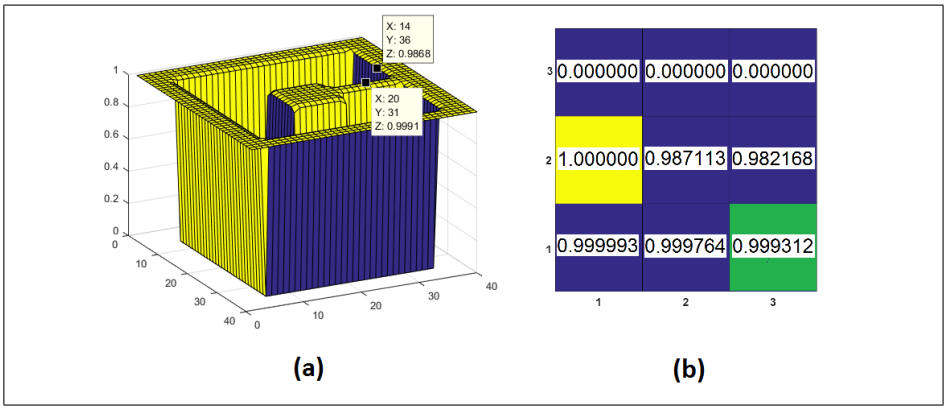

Fig. 8: Maximum probability of satisfying the LTL property in (a) Region 2 and (b) Region 3: The result generated by our proposed algorithm is identical to PRISM result

5.3 Simulation Results

The simulation parameters are set as $\mu = 0.75$ and $\gamma = 0.9$. Fig. 5 gives the results of the learning for the expression (9) in Region 1 and Region 2 after $400,000$ iterations and 200 learning episodes. According to (9) the robot has to

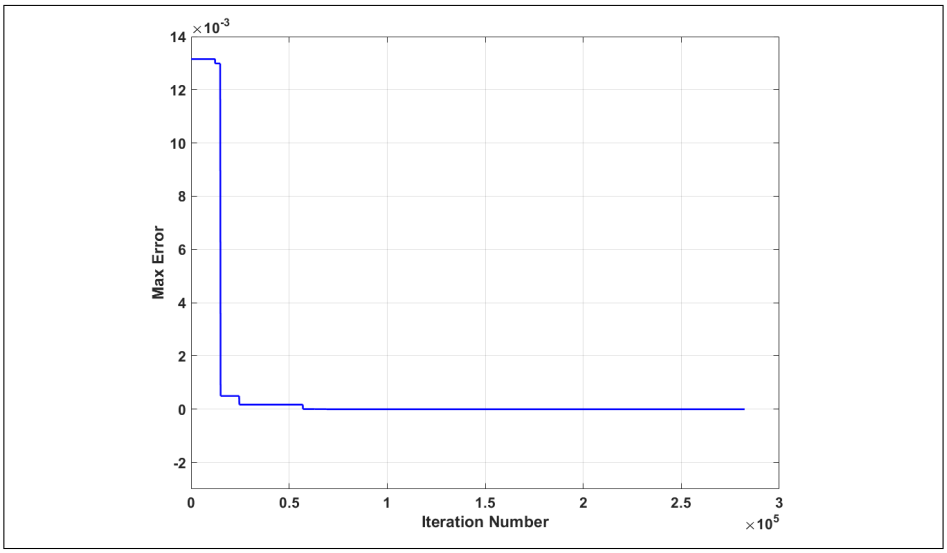

Fig. 9: Maximum error between PRISM results and our online algorithm in PSP^* calculation

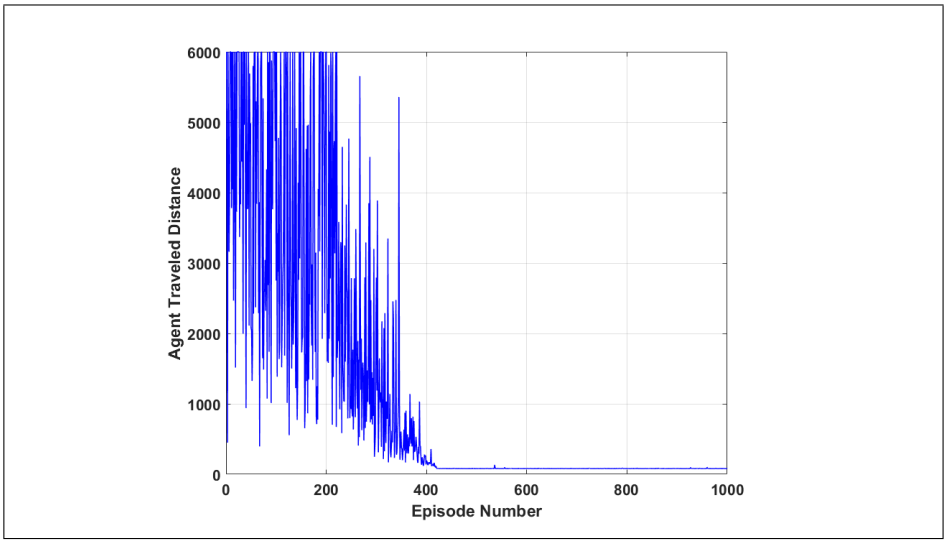

Fig. 10: The distance that agent traverses from initial state to final state

avoid the red (unsafe) regions, visit the light-blue (pre-target) area at least once and then go to the yellow (target) region. Recall that the robot desired action is executed with a probability of 85%. Thus, there might be some undesired deviations in the robot path.

Fig. 6 gives the results of the learning for the LTL formula (7) in Region 1 and Region 2 after $400,000$ iterations and 200 learning episodes. The intuition

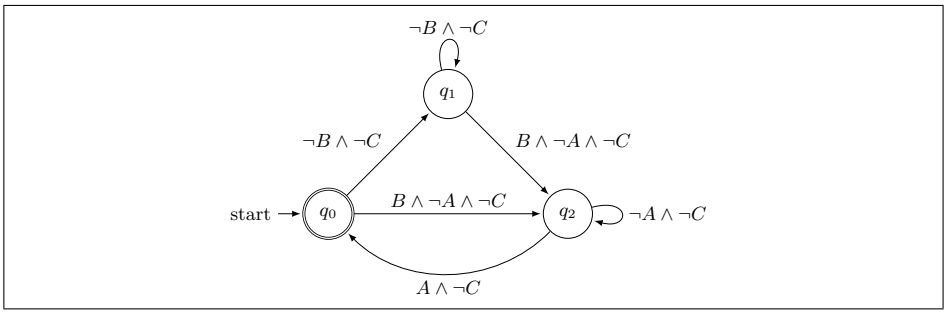

Fig. 11: LDBA expressing the LTL formula in (10) with removed transitions labelled $A \wedge B$ (since it is impossible to be at A and B at the same time)

behind the LTL formulae in (7) is that the robot has to avoid red (unsafe) areas until it reaches the yellow (target) region, otherwise the robot is going to be stuck in the red (unsafe) area.

Finally, in Fig. 7 the learner tries to satisfy the LTL formula $\Diamond\Box t$ in (8). The learning takes 1000 iterations and 20 learning episodes.

Fig. 8 gives the result of our proposed value iteration method for calculating the maximum probability of satisfying the LTL property in Region 2 and Region 3. In both cases our method was able to accurately calculate the maximum probability of satisfying the LTL property. We observed a monotonic decrease in the maximum error between the correct PSP^* calculated by PRISM and the probability calculation by our proposed algorithm (Fig. 9).

Fig. 10 shows the distance that agent traverses from initial state to final state at each learning episode in Region 1 under (9). After almost 400 episodes of learning the agent is converges to the final policy and the traveled distance stabilizes.

5.4 Comparison with a DRA-based algorithm

The problem of LTL-constrained RL is also investigated in [6], where the authors propose to translate the LTL property into a DRA and then to employ a product MDP. A 5×5 grid world is considered and staring from $(0, 3)$ the learner has to visit two regions infinitely often (areas A and B in Fig. 12). The learner has to also avoid the area C. The considered LTL property is the following formula:

$$\Box\Diamond A \wedge \Box\Diamond B \wedge \Box\neg C. \tag{10}$$

The product MDP in [6] contains 150 states, which means that the Rabin automaton has 6 states. Fig. 12-a shows the trajectories that are generated by the RL algorithm after 600 iterations. However, by employing our proposed algorithm we were able to generate the same trajectories with only 50 iterations (Fig. 12-b). The automaton that we considered is an LDBA with only 3 states as in Fig. 11. This explains why our algorithm found the trajectories faster as the learner has to explore only 75 states (in the product MDP). In addition to this, we proposed

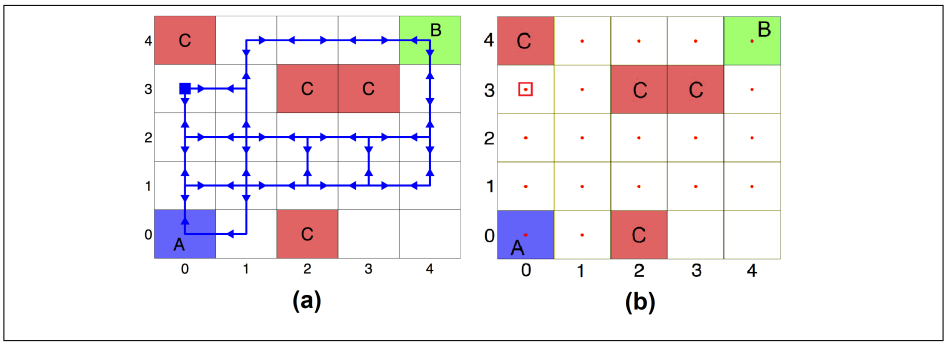

Fig. 12: (a) Example considered in [6] (b) Trajectories generated by our proposed algorithm in [6] example

a trajectory-based value iteration method to locally calculate the maximum probability of satisfying the LTL property. This allows us to quantitatively model check the underlying MDP.

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

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

Therefore for each $i \in \{1, \ldots, n_F\}$ a separate reward assignment is proposed in [6]. Clearly, the accepting condition of a Büchi automaton is much simpler than a Rabin automaton.

6 Conclusion

In this paper we proposed a method to constrain RL resulting traces by using an LTL property. We argued that converting the LTL property to an LDBA results in a significantly smaller automaton than DRA, which decreases the size of the product MDP and increases RL's convergence rate [14]. However, the standard approach in the literature is to convert the LTL property into a DRA. A DRA needs more complicated accepting conditions than a Büchi automaton; thus, a more complex reward assignment is needed. Therefore, in addition to the more succinct product MDP and faster convergence, our algorithm is easier to implement as opposed to standard methods that convert the LTL property to a DRA. Additionally, we proposed a value iteration method to calculate the probability of satisfaction of the LTL property. We argue that with this trajectory-based method we are able to apply our algorithm to large-scale MDPs which are hard for model-based methods to handle. Use of RL for policy generation allows the value iteration algorithm to focus on parts of state space that are relevant to the property. This results in a faster calculation of probability values when compared to DP, where these values are updated for the whole state space.

References

1. Puterman, M.L.: Markov decision processes: Discrete stochastic dynamic programming. John Wiley & Sons (2014)
2. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial intelligence **112**(1-2) (1999) 181–211
3. Rahili, S., Ren, W.: Game theory control solution for sensor coverage problem in unknown environment. In: Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on, IEEE (2014) 1173–1178
4. Hasanbeig, M., Pavel, L.: On synchronous binary log-linear learning and second order Q-learning. In: The 20th World Congress of the International Federation of Automatic Control (IFAC), IFAC (2017)
5. Hasanbeig, M.: Multi-agent learning in coverage control games. Master's thesis, University of Toronto (Canada) (2016)
6. Sadigh, D., Kim, E.S., Coogan, S., Sastry, S.S., Seshia, S.A.: A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In: Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on, IEEE (2014) 1091–1096
7. Abbeel, P., Coates, A., Quigley, M., Ng, A.Y.: An application of reinforcement learning to aerobatic helicopter flight. Advances in neural information processing systems **19** (2007) 1
8. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540) (2015) 529–533
9. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587) (2016) 484–489
10. van Otterlo, M., Wiering, M.: Reinforcement learning and Markov decision processes. In: Reinforcement Learning. Springer (2012) 3–42
11. Safra, S.: On the complexity of omega-automata. In: Foundations of Computer Science, 1988., 29th Annual Symposium on, IEEE (1988) 319–327
12. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Logic in Computer Science, 2006 21st Annual IEEE Symposium on, IEEE (2006) 255–264
13. Alur, R., La Torre, S.: Deterministic generators and games for LTL fragments. ACM Transactions on Computational Logic (TOCL) **5**(1) (2004) 1–25
14. Sickert, S., Esparza, J., Jaax, S., Křetínskỳ, J.: Limit-deterministic Büchi automata for linear temporal logic. In: International Conference on Computer Aided Verification, Springer (2016) 312–332
15. Tkachev, I., Mereacre, A., Katoen, J.P., Abate, A.: Quantitative model-checking of controlled discrete-time markov processes. Information and Computation **253** (2017) 1–35
16. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. Volume 1. MIT press Cambridge (1998)
17. Fu, J., Topcu, U.: Probably approximately correct MDP learning and control with temporal logic constraints. In: Robotics: Science and Systems X. (2014)
18. Wolff, E.M., Topcu, U., Murray, R.M.: Robust control of uncertain Markov decision processes with temporal logic specifications. In: Decision and Control (CDC), 2012 IEEE 51st Annual Conference on, IEEE (2012) 3372–3379

19. Svorenova, M., Cerna, I., Belta, C.: Optimal control of MDPs with temporal logic constraints. In: Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on, IEEE (2013) 3938–3943
20. Wen, M., Ehlers, R., Topcu, U.: Correct-by-synthesis reinforcement learning with temporal logic constraints. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, IEEE (2015) 4983–4990
21. Junges, S., Jansen, N., Dehnert, C., Topcu, U., Katoen, J.P.: Safety-constrained reinforcement learning for MDPs. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer (2016) 130–146
22. Li, X., Vasile, C.I., Belta, C.: Reinforcement learning with temporal logic rewards. arXiv preprint arXiv:1612.03471 (2016)
23. Ding, X.C., Pinto, A., Surana, A.: Strategic planning under uncertainties via constrained Markov decision processes. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on, IEEE (2013) 4568–4575
24. Lahijanian, M., Wasniewski, J., Andersson, S.B., Belta, C.: Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In: Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE (2010) 3227–3232
25. Pathak, S., Pulina, L., Tacchella, A.: Verification and repair of control policies for safe reinforcement learning. Applied Intelligence (2017) 1–23
26. Andersson, O., Heintz, F., Doherty, P.: Model-based reinforcement learning in continuous environments using real-time constrained optimization. In: AAAI. (2015) 2497–2503
27. Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínskỳ, J., Kwiatkowska, M., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: International Symposium on Automated Technology for Verification and Analysis, Springer (2014) 98–114
28. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. arXiv preprint arXiv:1708.08611 (2017)
29. Thomaz, A.L., Breazeal, C.: Teachable robots: Understanding human teaching behavior to build more effective robot learners. Artificial Intelligence **172**(6-7) (2008) 716–737
30. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4) (1992) 279–292
31. Sickert, S., Křetínskỳ, J.: MoChiBA: Probabilistic LTL model checking using limit-deterministic Büchi automata. In: International Symposium on Automated Technology for Verification and Analysis, Springer (2016) 130–137
32. Baier, C., Katoen, J.P., Larsen, K.G.: Principles of model checking. MIT press (2008)
33. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: International Conference on Computer Aided Verification, Springer (2011) 585–591
34. Ashok, P., Chatterjee, K., Daca, P., Křetínský, J., Meggendorfer, T. In: Value Iteration for Long-Run Average Reward in Markov Decision Processes. Springer International Publishing, Cham (2017) 201–221
35. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-dynamic programming: an overview. In: Decision and Control, 1995., Proceedings of the 34th IEEE Conference on. Volume 1., IEEE (1995) 560–564
36. Durrett, R.: Essentials of stochastic processes. Volume 1. Springer (1999)

Appendix 1

Theorem 2. Let MDP $\mathbf{M_N}$ be the product of an MDP $\mathbf{M}$ and an automaton $\mathbf{N}$ where $\mathbf{N}$ is the LDBA associated with the desired LTL property φ. Any optimal policy which optimizes the expected utility will satisfy the property.

Proof:
We first show that for any policy that satisfies the property the expected utility is positive. Assume that there exists a policy $\overline{Pol}$ that satisfies φ. Policy $\overline{Pol}$ induces a Markov chain $\mathbf{M}_{\mathbf{N}}^{\overline{Pol}}$ when it is applied over the MDP $\mathbf{M_N}$. This Markov chain is a disjoint union of a set of transient states $T_{\overline{Pol}}$ and n sets of irreducible recurrent classes $R^i_{\overline{Pol}}$ [36] as:

$$\mathbf{M}_{\mathbf{N}}^{\overline{Pol}} = T_{\overline{Pol}} \sqcup R^1_{\overline{Pol}} \sqcup ... \sqcup R^n_{\overline{Pol}}.$$

From (2), policy $\overline{Pol}$ satisfies φ if and only if:

$$\exists R^i_{\overline{Pol}} \text{ s.t. } \forall j \ F_j^{\otimes} \cap R^i_{\overline{Pol}} \neq \emptyset. \tag{11}$$

From the irreducibility of the recurrent class $R^i_{\overline{Pol}}$ we know that all the states in $R^i_{\overline{Pol}}$ communicate with each other. Therefore, they are surely reachable from each other in the MDP $\mathbf{M_N}$ and hence belong to the same MEC. From Definition 13 and considering the fact that $R^i_{\overline{Pol}}$ has non-empty intersection with all $F_j^{\otimes}$, we can conclude that the states of $R^i_{\overline{Pol}}$ belong to an AMEC. This means that in the recurrent class $R^i_{\overline{Pol}}$ the transition rewards are positive. From Definition 3, the expected utility for the initial state $\bar{s} \in \mathcal{S}^{\otimes}$ is:

$$U^{\overline{Pol}}(\bar{s}) = \mathbb{E}^{\overline{Pol}}[\sum_{n=0}^{\infty} \gamma^n \ R(S_n, \overline{Pol}(S_n))|S_0 = \bar{s}].$$

Since the reward for transition to AMEC is positive and $R^i_{\overline{Pol}}$ is a recurrent class, it follows that $U^{\overline{Pol}}(\bar{s}) > 0$.

In the following, by contradiction, we show that any optimal policy Pol^* which optimizes the expected utility will satisfy the property.
Contradiction Assumption: Suppose that the optimal policy Pol^* does not satisfy the property φ.
By this assumption:

$$\forall R^i_{Pol^*}, R^i_{Pol^*} \cap \text{AMECs} = \emptyset. \tag{12}$$

From (12) we can conclude that there is no state transition with positive reward in any $R^i_{Pol^*}$. Furthermore, the transient class T_{Pol^*} has no intersection with AMECs since with Pol^* there is always a positive probability for each state to be selected in an AMEC. From Definition 3, the expected utility for $\bar{s}$ is

$$U^{Pol^*}(\bar{s}) = \mathbb{E}^{Pol^*}[\sum_{n=0}^{\infty} \gamma^n \ R(S_n, Pol^*(S_n))|S_0 = \bar{s}].$$

This means that for $\bar{s}$ we have $U^{Pol^*}(\bar{s}) = 0$. However, as we have shown $U^{\overline{Pol}}(\bar{s}) > 0$. This contradicts the optimality of policy Pol^*. Thus, the contradiction assumption is false and Pol^* satisfies the property φ. ⌟

Appendix 2

Proposition 4. Our proposed value iteration update rule converges at least after N iterations where $N = [L/D] + 1$ and $L = ||PSP - PSP^*||_\infty$, $D = ||F\ PSP - PSP||_\infty$.

Proof:
Let PSP^* be the optimal value obtained by a model-based method. From Lemma 1:

$$||F(F\ PSP) - F\ PSP||_\infty \leq ||F\ PSP - PSP||_\infty = D.$$

This is true for all subsequent steps, i.e.

$$||F(F(F\ PSP)) - F(F\ PSP)||_\infty \leq ||F(F\ PSP) - F\ PSP||_\infty,$$

$$||(F^{(4)}PSP)) - (F^{(3)}PSP))||_\infty \leq ||(F^{(3)}PSP) - F^{(2)}PSP||_\infty,$$

$$\vdots$$

Thus, assuming that PSP converged to PSP^* at iteration N,

$$||(F^{(N)}PSP)) - (F^{(N-1)}PSP))||_\infty \leq ||(F\ PSP) - PSP||_\infty.$$

By knowing this, we can argue that at least $N = [L/D] + 1$ steps are needed to reach PSP^* starting from PSP. ⌟