Cautious Reinforcement Learning with Logical Constraints

Mohammadhosein Hasanbeig
University of Oxford
Oxford, UK
hosein.hasanbeig@cs.ox.ac.uk

Alessandro Abate
University of Oxford
Oxford, UK
alessandro.abate@cs.ox.ac.uk

Daniel Kroening
University of Oxford
Oxford, UK
kroening@cs.ox.ac.uk

ABSTRACT

This paper presents the concept of an adaptive *safe padding* that forces Reinforcement Learning (RL) to synthesise optimal control policies while ensuring safety during the learning process. Policies are synthesised to satisfy a goal, expressed as a temporal logic formula, with maximal probability. Enforcing the RL agent to stay safe during learning might limit the exploration, however we show that the proposed architecture is able to automatically handle the trade-off between efficient progress in exploration (towards goal satisfaction) and ensuring safety. Theoretical guarantees are available on the optimality of the synthesised policies and on the convergence of the learning algorithm. Experimental results are provided to showcase the performance of the proposed method.

ACM Reference Format:
Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2020. Cautious Reinforcement Learning with Logical Constraints. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Reinforcement Learning (RL) is an algorithm with no supervision that can be used to train an agent to interact with an unknown environment. The dynamics of the interaction are often assumed to be a Markov Decision Process (MDP). The key feature of RL is its sole dependence on a set of experiences, which are gathered by interacting with, i.e., *exploring*, the environment. This makes RL inherently different than classical dynamic programming methods [32] and automatic control approaches, in the sense that it can optimally solve the decision-making problem without any prior knowledge about the MDP model [36]. This very practical feature has paved the way for applications of RL in economics, engineering, and biology *inter alia*, to solve sequential decision-making problems when no model is available [1, 21–23, 26, 34, 43].

While the existing RL methods deliver good training outcomes, by and large they lack guarantees on what happens *during training*. Existing results rely either on "soft safety" or on "ergodicity" assumptions. The essence of soft safety is that unsafe states, which are to be avoided, may still be visited regardless of the consequent catastrophic outcome. The ergodicity assumption means that any state is eventually reachable from any other state if a proper policy is followed – this second assumption allows a (non-episodic) RL agent to explore by simply favouring states that have rarely been visited, even if they are unsafe in practice. These assumptions might be reasonable for certain applications, such as gaming or virtual environments, but are not affordable for many safety-critical physical systems, which may break before exploration completes. Thus, unsurprisingly, most of the existing exploration methods are impractical in safety-critical scenarios where the aforementioned assumptions do not hold.

Safe RL is an active area of research focusing on the training of agents with guarantees on safety [17]. However, most of these methods minimize the risk that the *trained agent* violates a safety specification, but do not ensure safety of exploration *during training* [2, 9, 11, 16, 18–20, 24, 28]. Recent approaches on this problem [8, 25, 27, 38, 41] are either computationally expensive or require explicit, strong assumptions about the model of agent-environment interactions.

In this work we take a step back from currently used exploration methods and recall that RL is originally inspired by cognitive and behavioural psychology [37]. When humans learn to control, they naturally account for what *they expect* to be safe, i.e., (1) they use their own a-priori prediction of the environment when choosing which behaviours to explore, and (2) they continuously update their knowledge and expectations using local observations. For example, when novice pilots learn to control a helicopter, they slowly pull the lever until the helicopter slightly lifts off the ground, then quickly land it back down. They will repeat this a few times, gradually increasing the time the helicopter hovers off the ground. At all times, they aim to prevent the likelihood of a disaster by ensuring that a safe landing is possible. In other words, they try to restrict exploration to a locally safe state-action region, which in this work we shall name *safe padding*. As their knowledge of the dynamics of the helicopter in its environment improves, they perform increasingly more sophisticated maneuvers, namely exploring new sequences of state-action pairs. It is interesting to notice that the maneuvers that a fully trained pilot will ultimately perform are initially incompatible with the safety of the learning agent, and as such might be located outside of the *safe padding* initially in use while learning.

Inspired by the cognitive approach to learning outlined above, we propose to equip the RL agent with a limited knowledge of its own dynamics, and with its local perception of safety. *Uncertain dynamics* characterise how the environment reacts to the actions of the agent. Much like a trainee pilot, the agent starts by performing exploratory cautious actions, and gradually, in line with the growing confidence about the environment obtained from observations, the range of acceptably safe actions grows, and the uncertain component of the dynamics becomes known.

Beyond the issue of safe exploration, in the RL literature task satisfaction is usually achieved by hand-engineering appropriate rewards [17]. In this context the difficulty is mapping complex, possibly long-term, sequential tasks to an appropriate reward structure [6]. If done incorrectly, the outcome of learning might be unexpectedly sub-optimal. As an extension of ongoing research

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand. © 2020 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

[20, 22], in this work we employ temporal logic, and more specifically Linear Temporal Logic (LTL) [31] as a formal reward shaping technique to specify task-related goals [10]. We convert a given LTL formula into an automaton that expresses the property [4], then translate the automaton into a state-adaptive reward structure. Using any off-the-shelf RL with the obtained reward structure results in policies that maximised the probability of verifying the given LTL formula. This framework, which we call Logically-Constrained RL (LCRL [20, 22]), is enhanced here by safe learning, with an architecture named *cautious RL*. Cautious RL is applicable to any standard reward-based RL.

While safety could be as well one of the tasks expressed in the LTL formula, meaning that the given LTL property will hold when deploying the trained agent, safety in the context of this work will be separately accounted for during training by means of safe padding. Technically, we propose a safe padding in combination with the state-adaptive reward function based on the task automaton over the state-action pairs of the MDP. Using this automatic reward shaping procedure, RL is able to generate a policy that satisfies the given task expressed as an LTL property with maximal probability, while the safe padding prevents violating safety during learning. Thus, the method we propose inherits aspects of reward engineering that are standard in RL, and at the same time infuses notions from formal methods that allow guiding the exploration safely, furthermore also certifying the learning outcomes in terms of the probability of satisfying the given task expressed as an LTL formula.

The proposed framework is related to, but cannot be reduced to, work on Constrained MDP (CMDP) [3], due to its generality and to its inherent structural differences: in this work LTL satisfaction is encoded into the expected return itself, while in CMDP algorithms the original objective is separate from the constraint. Focusing exclusively on the safety fragment of LTL, the concept of shielding is proposed in [2]: the proposed shield is a reactive machine that ensures that the agent remains safe during learning. To express the specification, [2] uses a DFA and then translates the problem into a safety game. This work has been extended to probabilistic CTL properties in [24], where a probabilistic model checking technique is used to construct the shield. Unlike this paper, in both [2, 24], the agent has to observe the entire MDP (and opponents) to construct a model of the safety game. [13, 15] address safety-critical settings in the context of cyber-physical systems, where the agent has to select a correct model within a heterogeneous set of models in model-based RL: [15] first generates a set of feasible models given an initial model and data on runs of the system. With such a set of feasible models, the agent has to learn how to safely identify which model is the most accurate; [14] further employs differential dynamic logic [30], a first-order logic for specifying and proving properties of hybrid models.

In summary, in this paper we contribute the following:

(1) Cautious RL: a safe exploration scheme for model-free RL. This is applicable to standard reward-based RL, however, we tailor it to the next goals:
(2) The use of LTL as task specification for policy synthesis in RL. Automatic reward shaping and task decomposition when the task is highly complex. Bringing (1) and (2) together, we obtain:
(3) Prediction of unsafe state-action pairs (safe padding) while learning and consequent limitation of exploration and of policy learning for LTL task satisfaction. The method guarantees asymptotic results.

2 BACKGROUND

2.1 Problem Setup

Definition 2.1 (Markov Decision Process, MDP). A finite MDP $\mathfrak{M}$ is a six tuple $(\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ where $\mathcal{S}$ is a finite set called the state space, $\mathcal{A}$ is a finite set of actions, s_0 is the initial state. $P(\cdot|s,a) \in \mathcal{P}(\mathcal{S})$ is the probability distribution over the next states given that action a has been taken in state s, where $\mathcal{P}(\mathcal{S})$ is the set of probability distributions on subsets of $\mathcal{S}$. $\mathcal{AP}$ is a finite set of atomic propositions and a labelling function $L : \mathcal{S} \to 2^{\mathcal{AP}}$ assigns to each state $s \in \mathcal{S}$ a set of atomic propositions $L(s) \subseteq 2^{\mathcal{AP}}$. □

A random variable $R(s,a) \sim \rho(\cdot|s,a) \in \mathcal{P}(\mathbb{R}^+)$ can be defined over the MDP $\mathfrak{M}$, representing the immediate reward obtained when action a is taken in a given state s where $\mathcal{P}(\mathbb{R}^+)$ is the set of probability distributions on subsets of $\mathbb{R}^+$, and ρ is the reward distribution. One possible realization of the immediate reward is denoted by $r(s,a)$.

Definition 2.2 (Stationary Deterministic Policy). A policy is a rule according to which the agent chooses its action at a given state. More formally, a policy π is a mapping from the state space to a distribution in $\mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ is the set of probability distributions on subsets of $\mathcal{A}$. A policy is stationary if $\pi(\cdot|s) \in \mathcal{P}(\mathcal{A})$ does not change over time and it is called a deterministic policy if $\pi(\cdot|s)$ is a degenerate distribution. □

An MDP controlled by a policy π induces a Markov chain $\mathfrak{M}^\pi$ with transition kernel $P^\pi(\cdot|s) = P(\cdot|s, \pi(s))$, and with reward distribution $\rho^\pi(\cdot|s) = \rho(\cdot|s, \pi(s))$ such that $R^\pi(s) \sim \rho^\pi(\cdot|s)$.

Definition 2.3 (Expected Discounted Return [36]). For any policy π on an MDP $\mathfrak{M}$, the expected discounted return in state s is

$$V^\pi_{\mathfrak{M}}(s) = \mathbb{E}^\pi[\sum_{n=0}^{\infty} \gamma^n \; r(s_n, a_n) | s_0 = s], \tag{1}$$

where $\mathbb{E}^\pi[\cdot]$ denotes the expected value by following policy π, γ is the discount factor, and $s_0, a_0, s_1, a_1 \ldots$ is the sequence of state-action pairs generated by policy π. The expected discounted return is often referred to as *value function*. Note that the discount factor γ is a hyper-parameter that has to be tuned. In particular, there is standard work in RL on state-dependent discount factors [29, 40, 42], which is shown to preserve convergence and optimality guarantees. Similarly, the action-value function is defined as:

$$Q^\pi_{\mathfrak{M}}(s,a) = \mathbb{E}^\pi[\sum_{n=0}^{\infty} \gamma^n \; r(s_n, a_n) | s_0 = s, a_0 = a]. \tag{2}$$

We drop the subscript $\mathfrak{M}$ when it is clear from the context. □

Definition 2.4 (Optimal Policy). An optimal policy π^* is defined as follows:

$$\pi^*(s) = \arg\sup_{\pi \in \varpi} V^\pi_{\mathfrak{M}}(s),$$

where ϖ is the set of stationary deterministic policies over $\mathcal{S}$. □

Theorem 2.5 ([7, 32]). *In an MDP $\mathfrak{M}$ with a bounded reward function and a finite action space optimal policies are stationary and deterministic.*

By Theorem 2.5, as long as the reward function is bounded and the action space is finite, the optimal policy in Definition 2.4 exists. This would be the case for this work.

2.2 Linear Temporal Logic

LTL formulae over a given set of atomic propositions $\mathcal{AP}$ are syntactically defined as [31]

$$\varphi ::= true \mid \alpha \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi \,\mathrm{U}\, \varphi, \tag{3}$$

where $\alpha \in \mathcal{AP}$, and the operators $\bigcirc$ and U are called *next* and *until*, respectively. Using the until operator we define two further temporal modalities: (1) eventually, $\Diamond\varphi = true \,\mathrm{U}\, \varphi$; and (2) always, $\Box\varphi = \neg\Diamond\neg\varphi$. An infinite *word* w over the alphabet $2^{\mathcal{AP}}$ in MDP $\mathfrak{M}$ is defined as an infinite sequence $w = l_0\, l_1\, l_2\, l_3 \ldots \in (2^{\mathcal{AP}})^\omega$, where ω denotes infinite repetition and $l_i \in 2^{\mathcal{AP}}$, $\forall i \in \mathbb{N}$. The language $\{w \in (2^{\mathcal{AP}})^\omega \text{ s.t. } w \models \varphi\}$ 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.

Definition 2.6 (Probability of Satisfying an LTL Formula). Starting from any state s and following a stationary deterministic policy π, we denote the probability of satisfying formula φ as $\mathbb{P}(s..^\pi \models \varphi)$, where $s..^\pi$ is the collection of all state sequences starting from s, generated under policy π. □

For any LTL property φ the set *Words*(φ) can be expressed by an LDBA. An LDBA is a special form of a Generalized Büchi Automaton (GBA) [33], defined as follows:

Definition 2.7 (Generalized Büchi Automaton). A GBA $\mathfrak{A} = (Q, q_0, \Delta, \Sigma, \mathcal{F})$ is a structure where Q is its finite set of states, $q_0 \in Q$ is the initial state, $\Delta : Q \times \Sigma \to 2^Q$ is a transition relation, $\Sigma = 2^{\mathcal{AP}}$ is a finite alphabet, and $\mathcal{F} = \{\mathcal{F}_1, ..., \mathcal{F}_f\}$ is the set of accepting conditions, where $\mathcal{F}_j \subseteq Q, 1 \leq j \leq f$. □

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

$$inf(\theta) \cap \mathcal{F}_j \neq \emptyset, \tag{4}$$

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

Definition 2.8 (LDBA). A GBA $\mathfrak{A} = (Q, q_0, \Delta, \Sigma, \mathcal{F})$ is limit-deterministic if Q can be partitioned into two disjoint sets $Q = Q_N \cup Q_D$ such that [33]:

- $\Delta(q, \alpha) \subset Q_D$ and $|\Delta(q, \alpha)| = 1$ for every state $q \in Q_D$ and for every $\alpha \in \Sigma$;
- for every $\mathcal{F}_j \in \mathcal{F}$, $\mathcal{F}_j \subseteq Q_D$; and
- $q_0 \in Q_N$, and all the transitions from Q_N to Q_D are non-deterministic ε-transitions[1]. □

[1] An ε-transition allows an automaton to change its state without reading any label.

Remark 2.1. *An LDBA is a GBA that has two partitions: one initial (Q_N) and one accepting (Q_D). The accepting part includes all the accepting states and has deterministic transitions. The LTL-to-LDBA construction used in this paper [33] results in an automaton with deterministic initial and accepting parts. According to Definition 2.8, the discussed structure is still an LDBA, though the determinism in the initial part is stronger than that required in the LDBA definition. We explain later why this matters for the proposed algorithm.*

Remark 2.2. *At any state q of an LDBA $\mathfrak{A}$, the output of the transition relation Δ is non-empty, namely all the states of $\mathfrak{A}$ are non-blocking. Further, any subset of Σ can be read at any state.* □

3 CAUTIOUS REINFORCEMENT LEARNING WITH LOGICAL CONSTRAINTS

3.1 Logically-guided Reinforcement Learning

In order to relate the structure of an MDP to that of an LDBA, for now we assume that the MDP graph and its transition probabilities are fully known. This allows us to formally define a synchronised structure that will be key for policy synthesis. This assumption is dropped entirely later, and we stress that policy synthesis can indeed be implemented on-the-fly over unknown MDPs via model-free RL.

Definition 3.1 (Product MDP). Given an MDP $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ and an LDBA $\mathfrak{A} = (Q, q_0, \Delta, \Sigma, \mathcal{F})$ with $\Sigma = 2^{\mathcal{AP}}$, the product MDP is defined as $\mathfrak{M} \otimes \mathfrak{A} = \mathfrak{P} = (\mathcal{S}^\otimes, \mathcal{A}^\otimes, s_0^\otimes, P^\otimes, \mathcal{AP}^\otimes, L^\otimes, \mathcal{F}^\otimes)$, where $\mathcal{S}^\otimes = \mathcal{S} \times Q$, $s_0^\otimes = (s_0, q_0)$, $\mathcal{AP}^\otimes = Q$, $L^\otimes : \mathcal{S}^\otimes \to 2^Q$ such that $L^\otimes(s, q) = q$ and $\mathcal{F}^\otimes \subseteq \mathcal{S}^\otimes$ is the set of accepting states $\mathcal{F}^\otimes = \{\mathcal{F}_1^\otimes, ..., \mathcal{F}_f^\otimes\}$, where $\mathcal{F}_j^\otimes = \mathcal{S} \times \mathcal{F}_j$. The transition kernel $P^\otimes(\cdot|s_i^\otimes, a) \in \mathcal{P}(\mathcal{S}^\otimes)$ is such that given the current state (s_i, q_i) and action a, the new state (s_j, q_j) is obtained such that $s_j \sim P(\cdot|s_i, a)$ and $q_j \in \Delta(q_i, L(s_j))$. In order to handle ε-transitions in $\mathfrak{A}$ we furthermore need to add the following transitions:

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

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

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

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

□

An example of a product MDP, as per Definition 3.1, is given in Fig. 1 for an instance of an MDP and for an LDBA generated from the LTL formula

$$\varphi = a \wedge \bigcirc(\Diamond\Box a \vee \Diamond\Box b).$$

Next, we propose a state-adaptive reward function based on the accepting condition of the automaton, so that maximisation of the expected cumulative reward (to be attained via RL) implies the maximisation of the satisfaction probability for the LTL formula (Definition 2.6).

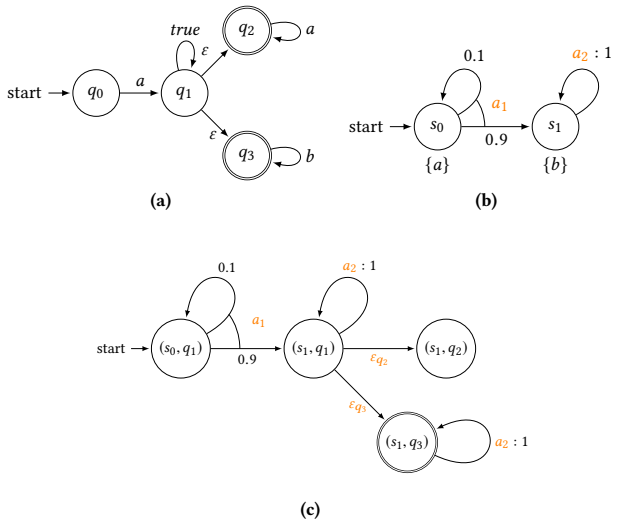


Figure 1: Example of product MDP: (a) the LDBA for $\varphi = a \wedge \bigcirc(\Diamond\Box a \vee \Diamond\Box b)$, (b) an instance MDP, and (c) the product according to Def. 3.1.

3.2 State-adaptive Reward

Before introducing the state-adaptive reward shaping scheme we need to provide a few definitions.

Definition 3.2 (Non-accepting Sink Component). A non-accepting sink component of an automaton, in this case an LDBA, $\mathfrak{A} = (Q, q_0, \Delta, \Sigma, \mathcal{F})$ is a directed graph induced by a set of states $Q_{sink} \subset Q$ such that (1) the graph is strongly connected; (2) it does not include all accepting sets $\mathcal{F}_k$, $k = 1, ..., f$; and (3) there exists no other strongly connected set $Q'_{sink} \subset Q$, $Q'_{sink} \neq Q_{sink}$ such that $Q_{sink} \subset Q'_{sink}$. We denote the union of all non-accepting sink components as Q_{sinks}. □

Remark 3.1. *Note that after taking a transition in the automaton that takes us to Q_{sinks}, it is not possible to satisfy the associated LTL property anymore, namely the probability of LTL satisfaction becomes zero under any policy. Identifying Q_{sinks} allows the agent to predict immediate labels that lead to a violation of the property. Thus, transitions $q \xrightarrow{\alpha \in \Sigma} q'$ to Q_{sinks} in the automaton are denoted by Δ_{sinks}.* □

Definition 3.3 (Accepting Frontier Function). Given an LDBA $\mathfrak{A} = (Q, q_0, \Delta, \Sigma, \mathcal{F})$, we define the accepting frontier function $AF : Q \times 2^Q \to 2^Q$, which executes the following operation over any given set $\mathbb{F} \in 2^Q$:

$$AF(q, \mathbb{F}) = \begin{cases} \mathbb{F} \setminus \mathcal{F}_j & : (q \in \mathcal{F}_j) \wedge (\mathbb{F} \neq \mathcal{F}_j) \\ \bigcup_{k=1}^{f} \mathcal{F}_k \setminus \mathcal{F}_j & : (q \in \mathcal{F}_j) \wedge (\mathbb{F} = \mathcal{F}_j). \end{cases} \tag{5}$$

□

Now assume that the agent is at state $s^\otimes = (s, q)$, takes action a and observes the subsequent state $s^{\otimes\prime} = (s', q')$. Note that since both the initial and accepting parts of the LDBA are deterministic, q' can be obtained on-the-fly[2]. The immediate reward is a scalar value, determined according to the following rule:

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

Here, $r_p > 0$ is a positive reward and $r_n = 0$ is a neutral reward. The set $\mathbb{A} \in 2^Q$ is called the *accepting frontier set*, and it is initialized as the family set of all accepting sets, i.e.,

$$\mathbb{A} = \{F_k\}_{k=1}^{f}. \tag{7}$$

The accepting frontier 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 before.

In short, after initialisation of $\mathbb{A} = \{F_k\}_{k=1}^{f}$ the accepting frontier function AF always excludes from $\mathbb{A}$ those accepting sets that have been visited or are being visited, unless it is the only remaining accepting set. In this case the accepting frontier $\mathbb{A}$ is reset, as per the second condition of (5). Thus, intuitively the set $\mathbb{A}$ always contains those accepting states that ought to be visited at any given time: in this sense the reward function is adapted to the accepting condition from the LDBA. The agent is guided by the above reward assignment to visit the accepting sets infinitely often, and consequently, to satisfy the given LTL property φ, as per (4). As in [20, 22], we will argue that we can learn an optimal policy generating traces that satisfy the given property φ with maximum probability.

Remark 3.2. *As in Definition 2.8 and Remark 2.1, the automaton transitions can be executed by reading the labels* only, *which makes the agent aware of the automaton state without explicitly constructing the product MDP. The transitions in the automaton can be executed on-the-fly as the agent reads the labels of the MDP states, without knowledge of the model structure or the transition probabilities (or their product). As such, our algorithm is implementable in a fully model free manner.* □

3.3 Safe Padding for Exploration

Ensuring safe exploration is critical when RL is employed to generate control policies in situations when learning from failure is unacceptable or extremely expensive, as in safety-critical autonomy for instance. We call this problem *safe policy synthesis*. We propose a *safe padding* for the agent by leveraging the agent limited knowledge about its own dynamics and its local perception of safety. Hence, the agent avoids violating the safety requirement (up to some probability level), while learning the optimal policy for task satisfaction.

Problem 1 (Safe Policy Synthesis). *Given an unknown black-box MDP $\mathfrak{M}$ and an LTL property φ a learning agent attains the following: (1) synthesises an optimal policy π^* via RL such that the induced Markov chain $\mathfrak{M}^{\pi^*}$ satisfies the LTL property with maximum possible probability; and (2) does not violate a safety requirement during learning. We assume that the transition probability function P in the MDP is only partly known. Technically, we assume that (i) the agent acquires prior knowledge about its own transition kernel (dynamics) $P_a : S \times A \times S \to [0, 1]$, that might not be accurate in the*

[2]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.

environment $\mathfrak{M}$ in which the agent operates. We also assume that (ii) the agent has a limited observability only *of the labelling function L in Definition 2.1: without knowing the full structure of the MDP, the agent is able to observe the labels of the surrounding states up to some distance from the current position.* □

Let us assume a starting belief about the agent transition kernel P_a, to be encoded as Dirichlet distributions [12, 35] via two functions $\Psi : \mathcal{S} \times \mathcal{A} \to \mathbb{N}$ and $\psi : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{N}$. Function $\psi(s, a, s')$ represents the number of times the agent executes action a in state s, thereafter moving to state s', and $\Psi(s, a) = \sum_{s' \in \mathcal{S}} \psi(s, a, s')$. The function $\Psi(s, a)$ is initialised to be one for every state-action pair, reflecting the fact that at any given state it is possible to take any action, and also avoiding division by zero; the function $\psi(s, a, s')$ is initialised to zero. Once the transition (s, a, s') is taken for the first time, $\Psi(s, a) \leftarrow 2$, so $\psi(s, a, s')$ has to be incremented to 2 to reflect the correct belief $P_a(s, a, s') = 1$ (Algorithm 1, lines 17–23).

The *safe padding* is a subset of the state-action space of the MDP that the agent considers safe to explore. As the agent explores and learns, the safe padding slowly expands, as much as in the flight control example the pilots slowly expands their comfort zone to learn how to control the helicopter. The expansion rate of the safe padding depends on how many times a particular state-action pair has been visited and on the corresponding update of the transition kernel P_a. The expansion mechanism and kernel update are explained shortly.

Let us recall that we have assumed that the agent has a limited observation range over the labels of the surrounding states (as per Problem 1). Assume that the observation radius is $r_o \geq 1$, meaning that the agent can *only* see the labels of states that have distance at most r_o over the MDP graph, i.e.

$$O(s) = \{s' \in \mathcal{S} \ \wedge \ d(s, s') \leq r_o\},$$

where $O(s) \subset \mathcal{S}$ is the set of states whose labels are visible at s, i.e., the agent current state, and $d(s, s')$ is the length of the shortest directed path from s to s'. Note that the agent can only see state labels, and has to rely on its current knowledge of the dynamics, namely P_a. Let the current state of the agent in the product MDP be $s^{\otimes} = (s, q)$. Define

$$O_{safe}(s) = \{x \in O(s), \ q \xrightarrow{L(x)} q' \notin \Delta_{sinks}\}, \tag{8}$$

as the set of safe states. Given the observation radius and the current belief of the agent about its own transition dynamics P_a, the agent performs a local, finite-horizon Bellman update over the observation area $O(s)$. For each state $x \in O_{safe}(s)$ and a horizon H the Bellman update is performed H times as follows [5]:

$$\begin{aligned} u_k(x) &= \min_{a \in \mathcal{A}} \sum_{x'} P_a(x, a, x') u_{k+1}(x'), \ k = H-1, \ldots, 0 \\ u_H(x) &= \mathbb{1}_{O_{safe}}(x), \end{aligned} \tag{9}$$

where x' is the subsequent state after taking action a in x, $u_k : \mathcal{S} \to [0, 1]$ is a local value function at time step k, and H is initialized as $H = r_o$. The value $u_H(x)$ is initialised as 1 if $x \in O_{safe}(s)$, and as 0 otherwise. With this initialisation, u_0 represents the agent estimation of the minimum probability of staying in O_{safe} within H steps, i.e. $u_0 = \mathbb{P}_{\min}(\Box^{\leq H} O_{safe})$. Notice that this estimation is indeed pessimistic and conservative. Hence, with a fixed horizon H the agent is able to calculate the maximum probability of violating the LTL property by picking action a at state $s^{\otimes}$ in H steps:

$$U_H(s^{\otimes}, a) = 1 - \sum_{s'} P_a(s, a, s') u_0(s'), \tag{10}$$

where $s^{\otimes} = (s, q)$. As assumed in Problem 1, the agent dynamics P_a is then updated considering the Maximum Likelihood Estimation (MLE) for the mean $P_a(s, a, s')$ [12, 35] as:

$$P_a(s, a, s') \leftarrow \frac{\psi(s, a, s')}{\Psi(s, a)}. \tag{11}$$

Here, $\psi(s, a, s')$ represents the number of times the agent executes action a in state s, thereafter moving to state s', whereas $\Psi(s, a) = \sum_{s' \in \mathcal{S}} \psi(s, a, s')$. Note that ψ and Ψ (and consequently P_a) are functions of the MDP state and action spaces, not of the product MDP, since they capture the agent dynamics over the original MDP, which remains the same regardless of the current automaton state q. Hence, the RHS of (10) only depends on state s and action a, and not the automaton state q.

Remark 3.3. *Note that for each state s, the Bellman updates in* (9) *are performed only over $O_{safe}(s)$. Recall that the set $O_{safe}(s)$ is the safe subset within the bounded observation area and thus, the computational burden of* (9) *is limited.* □

3.4 Safe Policy Synthesis with Logical Constraints

At this point we bring together the use of the safe padding with the constrained learning architecture generating policies that satisfy the given LTL formula. In order to pick the most optimal yet safe action at each state, we propose a *double learner* architecture, as explained in the following.

The first part is an *optimistic* learner that employs Q-learning (QL) [39] to maximize the expected cumulative return as in (2). For each state $s^{\otimes} \in \mathcal{S}^{\otimes}$ and for any action $a \in \mathcal{A}^{\otimes}$, QL assigns a quantitative value $Q : \mathcal{S}^{\otimes} \times \mathcal{A}^{\otimes} \to \mathbb{R}^+$, which is initialized with an arbitrary and finite value over all state-action pairs. The Q-function is updated by the following rule when the agent takes action a at state $s^{\otimes}$:

$$Q(s^{\otimes}, a) \leftarrow Q(s^{\otimes}, a) + \mu[R(s^{\otimes}, a) + \gamma \max_{a' \in \mathcal{A}^{\otimes}} (Q(s^{\otimes\prime}, a')) - Q(s^{\otimes}, a)], \tag{12}$$

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

Under mild assumptions over the learning rate, for finite-state and -action spaces QL converges to a unique limit, call it Q^* [39]. Once QL converges, the optimal policy $\pi^* : \mathcal{S}^{\otimes} \to \mathcal{A}^{\otimes}$ for $\mathfrak{P}$ can be generated by selecting the action that yields the highest Q^*, i.e.,

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

It has been shown [20, 22] that the optimal policy π^* generates traces that satisfy the given property φ with maximum probability.

Of course, adhering to the optimistic learner policy by no means guarantees to keep the agent safe *during* the exploration. This is

Algorithm 1: Cautious RL

input : LTL specification, *it_threshold*, γ, μ, r_o, $p_{critical}$, P_a
output : π^*

convert the desired LTL property to an equivalent LDBA $\mathfrak{A}$
initialize $\mathbb{A} = \{F_k\}_{k=1}^{f}$
initialize $\kappa = 1,\ \forall s \in \mathcal{S}$
initialize horizon $H = r_o,\ \forall s \in \mathcal{S}$
initialize $Q : \mathcal{S}^{\otimes} \times \mathcal{A}^{\otimes} \rightarrow \mathbb{R}^{+}$
initialize *episode-number* := 0
initialize *iteration-number* := 0
while Q *is not converged* **do**
  episode-number $\leftarrow$ *episode-number* + 1
  $s^{\otimes} = (s_0, q_0)$
  while $(q \notin \mathcal{Q}_{sink} :\ s^{\otimes} = (s, q)) \wedge$ (*iteration-number* < *it_threshold*) **do**
    iteration-number $\leftarrow$ *iteration-number* + 1
    # **pessimistic learner**
    calculate $U_H(s^{\otimes}, a)$ using P_a as in (10)
    generate $\mathcal{A}_p^H(s^{\otimes})$ as in (13)
    choose $a_* = \text{argmax}_{a \in \mathcal{A}_p^H[1:\kappa]}\ Q(s^{\otimes}, a) - r_p U_H(s^{\otimes}, a)$
    $\Psi(s^{\otimes}, a_*) \leftarrow \Psi(s^{\otimes}, a_*) + 1$
    execute action a_* and observe the next state $s_*^{\otimes}$
    if $\Psi(s^{\otimes}, a_*) = 2$ then
      $\psi(s^{\otimes}, a_*, s_*^{\otimes}) = 2$
    else
      $\psi(s^{\otimes}, a_*, s_*^{\otimes}) \leftarrow \psi(s^{\otimes}, a_*, s_*^{\otimes}) + 1$
    end
    update $P_a(s, a_*, s_*)$ as in (11)
    update $H(s)$
    update $\kappa(s)$
    # **optimistic learner**
    receive the reward $R(s^{\otimes}, a_*)$
    $\mathbb{A} \leftarrow Acc(s_*, \mathbb{A})$
    $Q(s^{\otimes}, a_*) \leftarrow$
     $Q(s^{\otimes}, a_*) + \mu[R(s^{\otimes}, a_*) - Q(s^{\otimes}, a_*) + \gamma \max_{a'}(Q(s_*^{\otimes}, a'))]$
    $s^{\otimes} = s_*^{\otimes}$
  end
end

where the second part of the architecture, i.e., the *pessimistic* learner, is needed: we exploit the concept of cautious learning and create a safe padding for the agent to explore safely. The pessimistic learner locally calculates $U_H(s^{\otimes}, a), \forall a \in \mathcal{A}^{\otimes}$ for a selected horizon H at the current state $s^{\otimes}$, and then outputs a set of permissive (safe) actions for the optimistic learner. Define a hyper-parameter $p_{critical}$ called *critical probability* to select actions $a \in \mathcal{A}^{\otimes}$. This is the probability that is considered to be critically risky (unsafe) and is defined prior to learning: any action a at state $s^{\otimes}$ that has $U_H(s^{\otimes}, a) \geq p_{critical}$ is considered as a critical action and has to be avoided. Accordingly, we introduce

$$\mathcal{A}_p^H(s^{\otimes}) = \{a \in \mathcal{A}^{\otimes} : U_H(s^{\otimes}, a) < p_{critical}\}. \tag{13}$$

This set is sorted over actions such that the first element has the lowest $U_H(s^{\otimes}, a)$ – with slight abuse of notations we write $\mathcal{A}_p^H[k]$ for the k-th element.

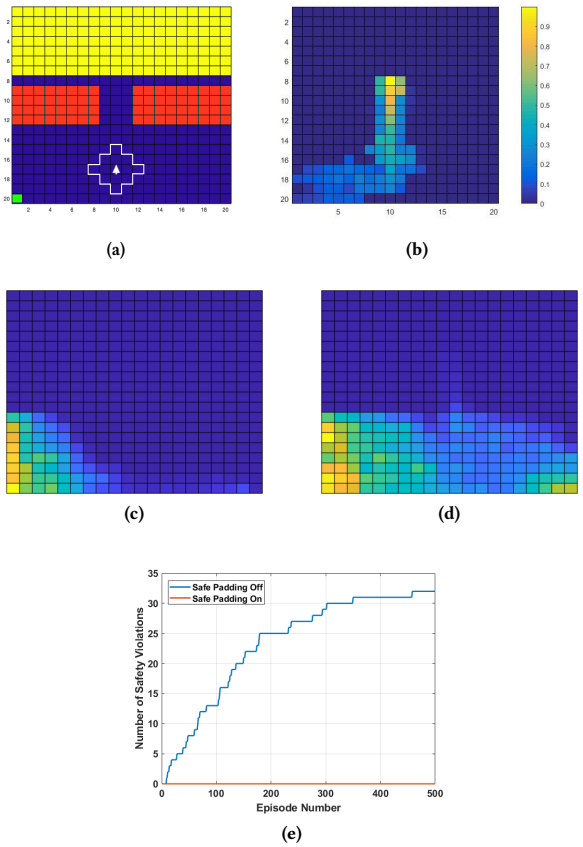


Figure 2: (a) slippery grid world and agent, represented by an arrow surrounded by an observation area. Labelling is yellow: *target*, red: *unsafe*, blue: *safe*, and green is the initial state s_0; (b) final value function $V(s)$, (c)–(d) state visitation number $v(s) = \sum_a \Psi(s, a)$ vs. time where the safe padding gradually grows and repels the agent from entering unsafe region; (e) number of times the agent reaches unsafe area (red) until RL converges with safe padding on vs. off.

At the beginning the pessimistic learner is conservative and only allows those actions in $\mathcal{A}_p^H$ that have index of less than κ, i.e., $\mathcal{A}_p^H[1:\kappa]$, where κ is a monotonically increasing function of the number of state visitations $v(s) = \sum_a \Psi(s, a)$, such that

$$\kappa(v)|_{v=1} = 1 \quad \text{and} \quad \lim_{v \rightarrow \infty} \kappa(v) = |\mathcal{A}_p^H|.$$

The horizon H follows the opposite rule, namely it is a monotonically decreasing function of $v(s)$ such that initially

$$H(v)|_{v=1} = r_o \quad \text{and} \quad \lim_{v \rightarrow \infty} H(v) = 1.$$

In other words, when the uncertainty around a state is high, the agent looks ahead as much as possible, i.e. $H = r_o$. Once the confidence level around that particular state increases then the agent considers riskier decisions by just looking one step ahead, i.e. $H = 1$. This essentially means that the safe padding grows as the uncertainty diminishes (or learning grows). Note that in practice, $\kappa(v)$ and $H(v)$ can be step-wise functions of v, and thus the agent is not

necessarily required to visit a state an infinite number of times to get to $H = 1$ and $\kappa = |\mathcal{A}_p^H|$.

Nevertheless, the infinite number of state (and action) visits is one of the theoretical assumptions of QL asymptotic convergence [39], which aligns with the proposed rate of change of κ and H. Owing to time-varying κ and H, when the agent synthesizes its policy, a subset of $\mathcal{A}^\otimes$ is only available, e.g., in the greedy case:

$$a^* = \underset{a \in \mathcal{A}_p^H[1:\kappa]}{\text{argmax}} \; Q(s^\otimes, a) - r_p U_H(s^\otimes, a),$$

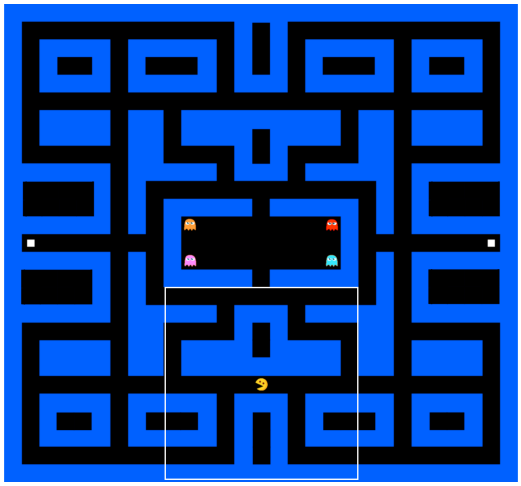
where the role of r_p is to balance Q and U_H. Note that since QL is an off-policy RL method, the choice of a^* during learning does not affect the convergence of the Q-function [39]. As the agent explores, the estimations of P_a, and thus of U_H, become more and more accurate, and the choice of actions become closer to optimal. Starting from its initial state s_0, the agent eventually expands the safe padding, i.e., the set of state-actions that it considers to be safe. The expansion occurs by diminishing the effect of the pessimistic learner, i.e., by decreasing the horizon H of $U_H(s^\otimes, a)$ and also by increasing κ in $\mathcal{A}_p^H(s^\otimes)$ until the effect of the pessimistic learner on decision making is minimal. Essentially, in the limit the role of the pessimistic agent is just to block the choice of actions that are critically unsafe according to $p_{critical}$ (actions that an optimal learned policy without the safe padding never takes, otherwise not optimal). However, the user-defined critical threshold $p_{critical}$ might affect the final policy of the agent in situations when acting safely may be at odds with acting optimally (Fig. 4).

4 EXPERIMENTS

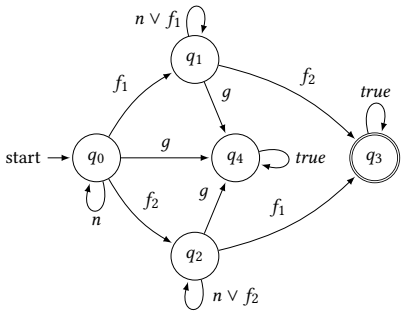
We consider numerical experiments that concern LTL-constrained safe control policy synthesis problems for a robot in a slippery grid-world and the classical Pacman game. In both experiments the agent has to experience risky situations in order to achieve the goal. This allows us evaluate the performance of the proposed safe padding architecture in protecting the agent from entering unsafe states.

For the robot example, let the grid be a 20×20 square over which the robot moves. In this setup, the robot location is the MDP state $s \in \mathcal{S}$. At each state $s \in \mathcal{S}$ the robot has a set of actions $\mathcal{A} = \{left, right, up, down, stay\}$ using which the robot is able to move to other states (e.g. s') with the probability of $P(s, a, s')$, $a \in \mathcal{A}$. At each state $s \in \mathcal{S}$, the actions available to the robot are either to move to a neighbour state $s' \in \mathcal{S}$ or to stay at the state s. In this example, 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.

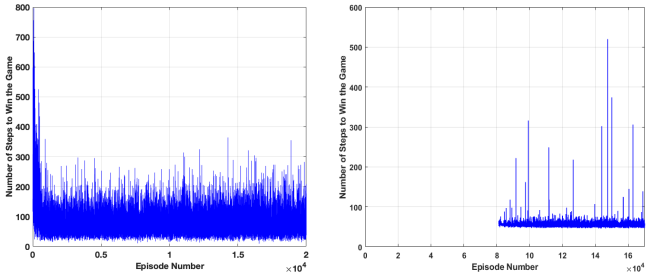
To get to the target state the agent has to cross a bridge (Fig. 2a) surrounded by unsafe states. The grid is slippery, namely from the agent's perspective, when it takes an action it usually moves to the intended cell, but there is an *unknown* probability that the agent is moved to a random neighbour cell. However, the agent prior belief P_a is that it can always move to the correct state and this is the dynamics known to the agent. The initial state of the agent is bottom left, $\gamma = 0.9$, $\mu = 0.85$, $p_{critical} = 0.82$, and the observation radius is $r_o = 2$. Note that for the sake of exposition, we intentionally picked $p_{critical} = 0.82$ close to the grid-world slipperiness probability of 0.85 and r_o close to the bridge gap, while in practice when the environment is unknown, $p_{critical}$ and r_o should be set conservatively.



(a)



(b)



(c) (d)

Figure 3: (a) Pacman environment with $|\mathcal{S}| > 80000$. Observation area is large square around initial condition. Square on the left is labelled as food 1 (f_1) and that the one on the right as food 2 (f_2), the state of being caught by a ghost is labelled as g, and the rest of the state space is neutral (n); (b) LDBA for the specification (15); (c) number of steps to complete the game with safe padding on (cautious RL), (d) and with safe padding off.

Table 1: Proportion of number of times that the agent ended in unsafe (fail) states, and proportion of number of times in which the agent finds a path satisfying the LTL specification during learning. Statistics are taken over 500 learning episodes in the slippery grid-world and over 20000 episodes in the Pacman experiment.

Case Study	Safe Padding	Fail Rate	Success Rate
Slippery Grid-world	Off	36.48%	63.52%
	On	0%	100%
Pacman	Off	52.69%	47.31%
	On	10.77%	89.23%

Similar to the pilot-helicopter example, the final goal of reaching the target is initially conflicting with the agent being safe since crossing the bridge has a high risk of slipping into an unsafe state. Thus, the agent has to slowly try different states while remaining safe, until it realises that there is no other way than crossing the high-risk bridge to achieve its goal. The LTL property associated with this task is as follows:

$$\Diamond target \wedge \Box \neg unsafe. \tag{14}$$

Notice that in this example the safety requirements we uphold while learning are embedded directly within the LTL formula for the task. In general the two requirements can be distinct.

To ensure the agent's safety, we create a safe padding based on the agent knowledge of its own dynamics. This safe padding gradually grows, allowing the agent to safely explore the MDP (Fig. 2c–d) while repelling the agent to get too close to unsafe areas. Thanks to this guarding effect of the safe padding, once the goal is reached, the agent can safely back-propagate the reward and shape the value function (Fig. 2b) according to which the safe policy is later generated. Furthermore, note that with Cautious RL the agent is focused on those parts of the state space that are most relevant to the satisfaction of the given LTL property.

There was no single incident of going to unsafe in this experiment even with such a limited observation radius (Table 1). With the safe padding on, training took 170 episodes for RL to converge and with safe padding off, it took 500 episodes.

The second experiment is the classic game Pacman, which is initialised in a tricky configuration likely to lead the agent to be caught by the roaming ghosts (Fig. 3a). In order to win the agent has to collect all tokens without being caught by ghosts:

$$\Diamond[(f_1 \wedge \Diamond f_2) \vee (f_2 \wedge \Diamond f_1)] \wedge \Box \neg g, \tag{15}$$

where the token on the left is labelled as f_1, the one on the right as f_2, and the state of being caught by a ghost is labelled as g. The constructed LDBA is shown in Fig. 3b. The ghost dynamics is stochastic: with a probability $p_g = 0.9$ each ghost is chasing Pacman (*chase mode*), else it executes a random move (*scatter mode*). Note that each combination of (Pacman, ghost1, ghost2, ghost3, ghost4) represents a state in the experiment, resulting in a state-space cardinality over 80000. As in the previous case study, here the safety requirements we uphold while learning are embedded directly within the LTL formula for the task.

Fig. 3c gives the results of learning with safe padding on and Fig. 3d off. Note that with the safe padding on, the agent was able

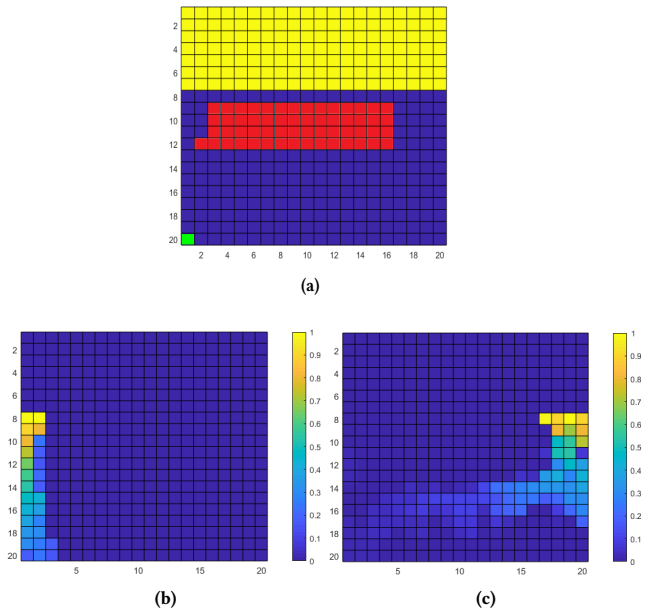


Figure 4: Safety and performance trade-off: (a) slippery grid world with two options to satisfy formula (14), where labelling is yellow: *target*, red: *unsafe*, blue: *safe*, and green is the initial state s_0; (b) value function $V(s)$ without safe padding; and (c) value function with safe padding (cautious RL).

to successfully escape the ghosts even from the beginning, with the cost of longer path to win whereas, without the safe padding it took 80000 episodes to score the very first win. In the Pacman experiment, the safe padding significantly reduced the number of times the agent got caught by the ghosts (Table 1).

5 CONCLUSIONS

In this paper, we have proposed *Cautious Reinforcement Learning*, a general method for safe exploration in RL usable on black-box MDPs, which ensures agent safety both during the learning process and for the final, trained agent. The proposed safe learning approach is in principle applicable to any standard reward-based RL. We have employed Linear Temporal Logic (LTL) to express an overall task (or goal), and to shape the reward for the agent in a provably-correct and safe scheme. We have proposed a double-agent RL architecture: one agent is pessimistic and limits the selection of the actions of the other agent, i.e., the optimistic one, which learns a policy that satisfies the LTL requirement. The pessimistic agent creates a continuously growing "safe padding" for the optimistic agent, which can learn the optimal task-satisfying policy, while staying safe during learning. The algorithm automatically manages the trade-off between the need for safety during the training and the need to explore the environment to achieve the LTL objective.

Acknowledgments. This work is in part supported by the HiClass project (113213), a partnership between the Aerospace Technology Institute (ATI), Department for Business, Energy & Industrial Strategy (BEIS) and Innovate UK.

REFERENCES

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. 2007. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems*. 1–8.

[2] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *AAAI*. 2669–2678.

[3] Eitan Altman. 1999. *Constrained Markov decision processes*. Vol. 7. CRC Press.

[4] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. 2008. *Principles of Model Checking*. MIT Press.

[5] Dimitri P Bertsekas and John N Tsitsiklis. 1996. *Neuro-dynamic Programming*. Vol. 1. Athena Scientific.

[6] Suda Bharadwaj, Stéphane Le Roux, Guillermo Pérez, and Ufuk Topcu. 2017. Reduction techniques for model checking and learning in MDPs. In *IJCAI*. 4273–4279.

[7] Rolando Cavazos-Cadena, Eugene A. Feinberg, and Raúl Montes-De-Oca. 2000. A note on the existence of optimal policies in total reward dynamic programs with compact action sets. *Mathematics of Operations Research* 25, 4 (2000), 657–666.

[8] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. 2019. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3387–3395.

[9] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. 2018. A lyapunov-based approach to safe reinforcement learning. In *Advances in neural information processing systems*. 8092–8101.

[10] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. 2018. *Model checking*. MIT Press.

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

[12] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1 (1977), 1–22.

[13] Nathan Fulton. 2018. *Verifiably Safe Autonomy for Cyber-Physical Systems*. Ph.D. Dissertation. Carnegie Mellon University Pittsburgh, PA.

[14] Nathan Fulton and André Platzer. 2018. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *AAAI*. 6485–6492.

[15] Nathan Fulton and Andre Platzer. 2019. Verifiably Safe Off-Model Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Vol. 11427. Springer, 413–430.

[16] Javier Garcia and Fernando Fernández. 2012. Safe exploration of state and action spaces in reinforcement learning. *JAIR* 45 (2012), 515–564.

[17] Javier Garcıa and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16, 1 (2015), 1437–1480.

[18] Peter Geibel and Fritz Wysotzki. 2005. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research* 24 (2005), 81–108.

[19] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2018. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099* (2018).

[20] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2019. Certified reinforcement learning with logic guidance. *arXiv preprint arXiv:1902.00778* (2019).

[21] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2019. Logically-Constrained Neural Fitted Q-Iteration. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2012–2014.

[22] Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J Pappas, and Insup Lee. 2019. Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees. In *Proceedings of the 58th Conference on Decision and Control*. IEEE, 5338–5343.

[23] Mohammadhosein Hasanbeig and Lacra Pavel. 2017. On Synchronous Binary Log-Linear Learning and Second Order Q-learning. In *The 20th World Congress of the International Federation of Automatic Control*. IFAC, 8987–8992.

[24] Nils Jansen, Bettina Könighofer, Sebastian Junges, and Roderick Bloem. 2018. Shielded decision-making in MDPs. *arXiv preprint arXiv:1807.06096* (2018).

[25] Xiao Li and Calin Belta. 2019. Temporal logic guided safe reinforcement learning using control barrier functions. *arXiv preprint arXiv:1903.09885* (2019).

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level Control Through Deep Reinforcement Learning. *Nature* 518, 7540 (2015), 529–533.

[27] Teodor Mihai Moldovan and Pieter Abbeel. 2012. Safe exploration in Markov decision processes. *arXiv preprint arXiv:1205.4810* (2012).

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

[29] Silviu Pitis. 2019. Rethinking the discount factor in reinforcement learning: A decision theoretic approach. *arXiv preprint arXiv:1902.02893* (2019).

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

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

[32] Martin L. Puterman. 2014. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons.

[33] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínskỳ. 2016. Limit-deterministic Büchi automata for linear temporal logic. In *CAV*. Springer, 312–332.

[34] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[35] Alexander L Strehl, Lihong Li, and Michael L Littman. 2009. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research* 10, Nov (2009), 2413–2444.

[36] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. Vol. 1. MIT Press Cambridge.

[37] Edward L. Thorndike. 1932. *The Fundamentals of Learning*. Teachers College Bureau of Publications.

[38] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. 2016. Safe exploration in finite Markov decision processes with Gaussian processes. In *Advances in Neural Information Processing Systems*. 4312–4320.

[39] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3-4 (1992), 279–292.

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

[41] Min Wen and Ufuk Topcu. 2018. Constrained cross-entropy method for safe reinforcement learning. In *Advances in Neural Information Processing Systems*. 7450–7460.

[42] Naoto Yoshida, Eiji Uchibe, and Kenji Doya. 2013. Reinforcement learning with state-dependent discount factor. In *ICDL*. IEEE, 1–6.

[43] Zhenpeng Zhou, Xiaocheng Li, and Richard N. Zare. 2017. Optimizing chemical reactions with deep reinforcement learning. *ACS Central Science* 3, 12 (2017), 1337–1344.