Safe and Certified Reinforcement Learning with Logical Constraints

Mohammadhosein Hasanbeig

Oriel College

University of Oxford

This thesis is submitted to the Computer Science Department of the University of Oxford, in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Trinity 2020

To Tala
the nature of daylight

Acknowledgements

First and foremost, I would like to thank my advisors Alessandro Abate, and Daniel Kroening for their dedication, support, and guidance. I have been extremely fortunate to have them as my advisors, and I would like to express my deepest gratitude for their trust, advice, and many thoughtful exchanges of ideas.

I would like to extend my sincere gratitude to my internal assessors Jan Calliess, Yarin Gal, Nick Hawes, Joel Ouaknine, Shimon Whiteson, and also my external examiner Krishnamurthy Dvijotham for their constructive and thorough feedback. Thanks to my college advisor, John Huber, and the college provost Moira Wallace for the delightful and supportive meetings during my stay in Oxford. I would like to thank Oriel College for the generous support and the stimulating environment it promotes, which made my Oxford experience truly unforgettable.

I am also very grateful to all my wonderful friends in the UK, US, Canada, and Iran, some even though far away, I know their support was there. This was indeed invaluable to me. In the end, I would like to thank my family for their never-ending unconditional love and support. I will forever be grateful.

Abstract

Reinforcement Learning (RL) is a widely employed machine learning architecture that has been applied to a variety of decision-making problems, from resource management to robot locomotion, from recommendation systems to systems biology, and from traffic control to superhuman performance in video games. However, RL has experienced limited success beyond rigidly controlled or constrained applications, and successful employment of RL in safety-critical scenarios is yet to be achieved. A principal reason for this limitation is the lack of a systematic formal approach to specify requirements as tasks (or learning constraints), and to provide guarantees with respect to these requirements and constraints, during and after learning. This work addresses these issues by proposing a general method that leverages the success of RL in learning high-performance controllers, while developing the required assumptions and theories for the satisfaction of formal requirements and guiding the learning process within safe configurations. Given the expressive power of Linear Temporal Logic (LTL) in formulating control specifications, we propose the first model-free RL scheme to synthesise policies for unknown black-box Markov Decision Processes (MDPs), where the reward function is formally shaped by an LTL specification. We convert the given property into a Limit Deterministic Büchi Automaton (LDBA), namely a finite-state machine expressing the property. Further, by exploiting the structure of the LDBA, we shape a synchronous reward function on-the-fly, and we discuss the assumptions under which RL is guaranteed to synthesise control policies whose traces satisfy the LTL specification with maximum probability possible. This probability (certificate) can also be calculated in parallel with policy learning when the state space of the MDP is finite: as such, the RL algorithm produces a policy that is certified with respect to the property. We also show that our method produces "best available" control policies when the logical property cannot be satisfied. The performance of the proposed method is evaluated via a set of numerical examples and benchmarks, where we observe an improvement of one order of magnitude in the number of iterations required for the policy synthesis, compared to existing approaches whenever available.

Statement of Originality

The contents of this dissertation are partly based on our publications at the *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS), 2019* [1], the *IEEE Conference on Decision and Control (CDC), 2019* [2], the *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS), 2020* [3], *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), 2020* [4], *AAAI Conference on Artificial Intelligence (AAAI), 2021* [5], the *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS), 2021* [6], *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021* [7, 8], the *CAV Workshop on Synthesis (SYNT)* [9, 10], and the *Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY)* [11]. At the time of writing, a number of chapters were under review in several conferences and journals, and a number of chapters were yet to be submitted for publication [12].

Contents

List of Figures

List of Tables

List of Abbreviations

AVI	Asynchronous Value Iteration, page 64
CEGIS	Counter-example Guided Inductive Synthesis, page 143
CMDP	Constrained MDP, page 17
DDPG	Deep DPG, page 105
DFA	Deterministic Finite Automaton, page 15
DPG	Deterministic Policy Gradient, page 37
DP	Dynamic Programming, page 2
DQN	Deep Q-Network, page 5
DRA	Deterministic Rabin Automaton, page 9
FVI	Fitted Value Iteration, page 93
GBA	Generalised Büchi Automaton, page 40
LCNFQ	Logically-Constrained NFQ, page 86
LCQL	Logically-Constrained QL, page 61
LCRL	Logically-Constrained Reinforcement Learning, page 12
LDBA	Limit Deterministic Büchi Automaton, page 41
LTL	Linear Temporal Logic, page 38
MDP	Markov Decision Process, page 30
NFQ	Neural Fitted Q-iteration, page 86
PAC MDP	Probably Approximately Correct MDP, page 12
POMDP	Partially Observable Markov Decision Process, page 142
QL	Q-learning, page 33

RL Reinforcement Learning, page 1

RSL Recurring Slope Lineae, page 110

SMC Statistical Model Checking, page 18

SMT Satisfiability Modulo Theories, page 12

SWA Stochastic Weight Averaging, page 118

VI Value Iteration, page 12

VQ Voronoi Quantiser, page 91

Chapter 1

Introduction

With the rise of artificial neural networks, Machine Learning (ML) has become ever more attractive and now provides solutions for a wide range of problems. Reinforcement Learning (RL), as a distinguished branch of ML, has similarly seen growing use and increased success. RL has achieved impressive results over the past few years, but often the learned solution is difficult to understand and examine by humans. Therefore, the robustness and reproducibility of RL results are usually evaluated by statistical testing and there is no systematic method to guarantee that a solution synthesised using RL meets the expectations of the designer of the algorithm. This becomes a pressing issue when applying RL to safety-critical systems, and there is a need for RL algorithms that are designed to train models with safety guarantees.

RL, in its simplest form, provides a solution to decision-making problems either when no prior knowledge is available or when analytical solutions are hard to be found [14]. More specifically, analytical methods developed in classic control theory,

e.g. robust control and adaptive control, provide optimal controllers that ensure safety and stability, but only when appropriate expert knowledge and an accurate model of the problem are available. Meanwhile, RL-based control approaches proved to be efficient when dealing with control problems with inaccurate and unknown models. This success is owed to RL being able to infer unknown models from data and actively improve the performance of the controller with a learned model, or entirely with a model-free scheme. Compared to classic control, RL requires no expert knowledge and considerably fewer assumptions about the problem. This practical approach has paved the way for RL to be employed in modern automatic control, game theory, economics, and biology inter alia, e.g. [13, 15–28], to solve sequential decision-making problems.

To formally model sequential decision-making problems, Markov Decision Processes (MDPs), a family of stochastic processes, are widely adopted in RL [29]. In an MDP, by choosing relevant actions over a set of states, a decision-maker (or an agent) can move probabilistically between the states [30] and receives a scalar reward. The outcomes from taking actions are in general probabilistic and not fully under the control of the agent [31]. An MDP is said to be solved when at any given state the agent is able to choose the most favourable actions so that the accrued reward is expected to be maximum in the long run: in other words, the goal is to find an optimal action-selection policy that returns the maximum expected reward [30].

When state and action spaces are finite, the stochastic behaviour of an MDP can be described by a transition probability matrix, which represents its mathematical model. In the case when the probability matrix is available, the traditional method to solve a given MDP is to use Dynamic Programming (DP). DP iteratively applies a Bellman operation on a value function expressing the expected reward of interest, which is defined over the "entire state space" of the MDP [31]. In this sense, DP is

an exact algorithm that provides analytic insight into the problem of sequential decision-making [32]. Most importantly, we understand the convergence properties of RL in relation to the optimal value function produced by DP [33, 34], as RL and DP share the same underlying formalism. When the state and action spaces are not finite, approximate DP is often employed. This approximation can be achieved by generating an abstract model of the MDP itself [35–38], or by inferring a (non-linear) regression model over the value function [39–41].

In practice, however, holding full knowledge of the internal stochastic dynamics of the problem MDP is often infeasible, and due to the DP reliance on the full state space and its known computational costs, the applicability of DP can be practically quite limited. Consider a robotic motion planning problem as an MDP: at a given state taking a particular action might move the robot to a different state each time due to several factors both in the environment where the robot operates and also in the mechanics of the robot; thus, in this example the MDP states are observable but transition probabilities can be unknown. In these scenarios, classical DP is of limited utility, because of its assumption of a perfectly-known MDP [14]. RL, on the other hand, solely depends on a set of experience samples gathered by exploring the MDP, which simplifies the Bellman's backup, and at the same time avoids the exhaustive sweeps over the whole state space.

Learning by collecting experience in RL is generally accomplished via two different methods: model-based learning and model-free learning. Model-based RL attempts to first model the MDP structure and its transition kernel, and then based on the built model it synthesises an optimal policy via DP or other planning algorithms. Alternatively, model-free RL learns an optimal policy directly by mapping state-action pairs to their expected reward, without the need for a model. Model-free RL is proved to converge to the same action selection policy as DP (over the original MDP) under mild assumptions [31, 33]. In this sense, model-free

methods are very accessible, as the agent directly learns an optimal policy without a need for an intermediate model.

Further to this, there exists a significant number of works from both experimental recordings in monkeys [42, 43], and human fMRI [44] that suggest the mammalian central nervous system exhibits a response pattern similar to model-free learning. Model-based learning methods, on the other hand, are not as general as model-free methods [45], even though they have convenient theoretical guarantees [46, 47]. Nevertheless, both methods are extensively used in a variety of applications such as robotics [17, 23, 24, 28], resource management [18, 21, 27], traffic management [13] and flight control [16], chemistry [19], and game play [15, 20].

Model-free or model-based, classical RL is focused on problems where the sets of states and actions in the MDP are finite. Nonetheless, many interesting real-world control tasks require actions to be taken in response to high-dimensional or real-valued sensory inputs [48]. As an example, consider the problem of drone control in which the drone state is represented as its Euclidean position $(x, y, z) \in \mathbb{R}^3$: the physical space of an MDP modelling the stochastic behaviour of the drone is uncountably infinite, namely continuous.

A straightforward technique to solve an (uncountably) infinite-state-action MDP with RL is to discretise the state and action spaces and to resort to conventional RL to find an approximate optimal policy [41]. However, the resulting discrete model can be often inaccurate and may not capture the full dynamics of the original MDP, leading to a sub-optimal policy synthesis. One might argue that by increasing the number of discrete states and actions this problem can be mitigated, however, the larger the number of discrete states and actions, the more expensive and time-consuming the learning process is. Thus, MDP discretisation has to always deal with the trade-off between accuracy and the curse of dimensionality.

A more elaborate solution is to gather a set of experience samples and then

use an approximation function constructed via regression from the samples set. A number of methods are available to approximate the expected reward function, e.g. sparse coarse coding [49], kernel-based modelling [50], tree-based regression [51], basis functions [52], etc. Among these methods, neural networks offer great promise in approximating the expected reward, due to their ability to generalise [53], and as a result there exist numerous successful applications of neural networks in RL for uncountably infinite or very large MDPs, e.g. TD-Gammon [54], Asynchronous RL [55], Neural Fitted Q-iteration (NFQ) [56], CACLA [57] and Deep Q-Network (DQN) [15].

Deep RL is arguably one of the recent breakthroughs in RL, whereby human-level performance has been achieved on a number of Atari games [15] by incorporating deep neural networks into RL, which laid the foundation for later successes in StarCraft [26], and the game of Go [58]. Deep-RL-based algorithms are often general enough so that the rules of different games do not have to be explicitly encoded for the agents to learn successful control policies, but at the price of very high sample complexity.

The success of deep RL has resulted in the extensive use of RL beyond small-scale classical contexts [26, 58–60]. In particular, employment of RL in safety-critical problems has recently been investigated [10, 12, 61–66], including autonomous driving [67, 68] and avionics [16, 69]. This however inevitably entails the need for correct-by-design policy synthesis, in order to guarantee, among other quantitative requirements, the safety of policies synthesised via RL. Furthermore, and with a different perspective on the notion, safety has to be present also "during learning", namely as the agent explores the MDP.

Existing exploration methods provide promising guarantees, though they tend to rely either on an ergodicity assumption or a soft safety assumption. The essence of soft safety is that unsafe states, which might be absorbing sink states, can be

visited regardless of its possible catastrophic outcome. For most physical systems this assumption is not affordable as these systems may break before any meaningful exploration happens. Alternatively, the ergodicity assumption requires that any state is eventually reached from any other state when a proper policy is followed. This assumption allows RL to explore by simply favouring states that have rarely been visited, and are potentially unsafe in practice. Thus, unsurprisingly in safety-critical scenarios when the aforementioned assumptions do not hold most of the exploration methods are unrealistic.

Further to the exploration concerns, the expression of complex requirements in RL has mostly relied on heuristics and reward engineering [70]: it often requires tedious parameter tuning to map complex goals to an appropriate reward structure in many problems [71, 72]. As a result, the agent may learn behaviours that are unpredictable or unfamiliar to humans. This is crucial in safety-critical applications, or when the agent operates in the proximity of humans. Whether or not the desired behaviour can be correctly embedded in the reward function can significantly impact the learning outcome. Particularly, the lack of provably-correct reward shaping can be a crippling issue for the employment of RL and deep RL in a large number of problems.

From a different perspective, despite the great success and generality of deep RL, it is not a natural representation of how humans perceive real-world tasks and also video games. In this work, we take a step back and would like to recall that humans already have prior high-level knowledge and associations regarding many elements that they see in the environment or appear on-screen in video games, e.g. "keys open doors". Given the useful domain knowledge that human experts can offer, and the otherwise huge challenge posed by randomly and exhaustively exploring large state spaces, new approaches have arisen that intend to combine human domain knowledge and insight with the ability of RL to eventually converge

to near-optimal policies. These include apprenticeship learning, imitation learning, and expert demonstrations [73–81], and have already shown great improvements over the state-of-the-art learning methods. However, these approaches are very much biased towards human behaviour and might not be able to find a globally optimal control policy when the human teacher believes that a local optimum is actually global. In other words, these approaches cannot be proven to correctly embed the desired optimal behaviour in the reward function. Therefore, introducing useful associations to RL in a formal provably-correct way allows the agent to lift its initial knowledge about the problem and to efficiently find the globally optimal policy, while avoiding an exhaustive exploration in the beginning or being biased towards human beliefs.

In this work, we address the aforementioned issues by leveraging concepts and techniques from Formal Methods. In particular, we employ Linear Temporal Logic (LTL) as a formal high-level language in which a complex task can be easily described [82], and later automatically shapes a provably-correct reward function that is directly fed to a model-free RL algorithm. The notion of safety in this sense is encompassed in the LTL specification and can be seen as RL generally not violating the property during and after learning. Thus, the proposed method benefits from reward engineering aspects that are standard in (safe) RL, and at the same time infuses notions from formal methods that allows guiding exploration and certifying learning outcomes in terms of the probability of staying safe. In addition to this algorithmic reward shaping, standard exploration schemes in RL, as pointed out before, impose unrealistic assumptions on ergodicity and soft safety. In this work, we propose an adaptive safe padding mechanism [3] that does not rely on the aforementioned assumptions while automatically balances the trade-off between efficient exploration and ensuring safety during learning.

1.1 Contributions

LTL [83] is a formal language that allows to express formal engineering requirements and specifications, to the extent that there exists a substantial body of research on the extraction of LTL properties from natural language sentences, e.g. [84–86]. LTL can express time-dependent properties, such as safety and liveness, and further allows to specify complex (e.g. repeating, sequential, conditional) tasks.

We propose the first model-free RL method, which allows to synthesise control policies under LTL specifications for a continuous-state continuous-action MDP (and its simpler, finite-state and finite-action case). We discuss the assumptions and a provably-correct reward shaping under which RL is guaranteed to synthesise control policies whose traces satisfy the LTL specification with maximum probability possible. In particular, the problem of "policy synthesis" is separated from (and does not depend on) that of "model learning" and can be addressed directly via model-free RL. This is not the case in previously introduced LTL synthesis algorithms, all of which rely on the process of (1) first learning a model of the MDP and then (2) finding an optimal policy based on the results of (1). We skip (1) and directly perform policy synthesis in (2) with the same guarantees as (1), when available. As we demonstrate in experiments, this drastically increases the convergence speed to the optimal policy and the scalability of the proposed scheme over standard methods. Additionally, in the case of continuous-state MDPs, and also continuous-state continuous-action MDPs, we introduced the first RL policy synthesis algorithm for LTL. The proposed methods solely rely on random experience samples gathered from the MDP and treat the MDP as a black-box.

From a learning perspective, by employing LTL in an RL context, we can infuse structural knowledge into the learning procedure, whilst avoiding the bias otherwise introduced by a human teacher, as discussed above. An LTL property is a formal,

un-grounded, and symbolic representation of the task and of its components. This allows the expression of complex properties and as we show later it can be extended to sub-task decomposition and hierarchical learning.

In the RL literature, the temporal dependency of rewards is often tackled with options [14], or, in general, the dependencies are structured hierarchically. Current approaches to hierarchical RL very much depend on state representations and whether they are structured enough for a suitable reward signal to be effectively engineered manually. Hierarchical RL therefore often requires detailed supervision in the form of explicitly specified high-level actions or intermediate supervisory signals [87–92]. On the contrary, in this work, a complex task is expressed by an LTL property where each component of the LTL property systematically structures the complex mission task into low-level, achievable task "modules" without requiring any supervision. The LTL property essentially acts as a high-level unsupervised guide for the agent, whereas the low-level planning is handled by a (deep) RL architecture. In particular, in order to show the enhancement of learning and also the scalability of the proposed architecture, we have picked the Atari 2600 game "Montezuma's Revenge" as one of our case studies, which is a well-known hard-to-solve problem in deep RL.

In order to synchronise LTL with RL, we convert the LTL property into an automaton, namely a finite-state machine [93]. In general, however, LTL-to-automaton translation may result in a non-deterministic model, over which policy synthesis for MDPs is in general not semantically meaningful. A standard solution to this issue is to use Safra construction to determinise the automaton, which as expected can increase its size dramatically [94, 95]. An alternative solution is to directly convert the given LTL formula into a Deterministic Rabin Automaton (DRA), which by definition rules out non-determinism. Nevertheless, it is known that such conversion results, in the worst case, in automata that are doubly

exponential in the size of the original LTL formula [96].

Conversely, in this work, we propose to express the given LTL property as a Limit Deterministic Büchi Automaton (LDBA) [97]. It is shown that this construction results in an exponential-sized automaton for LTL\GU[1], and it results in nearly the same size as a DRA for the rest of LTL. As we show later, this succinctness significantly impacts the convergence speed and sample efficiency of the learning algorithm. However, translation of LTL to LDBA introduces non-trivial problems into the learning process, such as partial non-determinism which are addressed in this work. Further to the discussion on succinctness, acceptance condition in LDBAs construction is, in general, easier to evaluate than DRAs, which makes policy synthesis algorithms much simpler to implement [98, 99].

We should emphasise that there exist a few LDBA construction algorithms for LTL, but not all of the resulting LDBAs can be employed for quantitative model-checking and probabilistic certification [100]. More importantly, the unique succinctness of the LDBA construction [97] used in our work is due to its "generalised Büchi" accepting condition, and other non-generalised LDBA constructions are inevitably larger. As shown later, a larger automaton negatively affects the convergence speed and sample efficiency of the learning algorithm.

Once the LDBA is generated from the given LTL property, we construct on-the-fly[2] a product between the MDP and the resulting LDBA, and then define a reward function that is synchronous with the accepting condition of the generalised Büchi automaton over the state-action pairs of the MDP. Using this algorithmic reward shaping procedure, RL is able to generate an optimal policy that returns the maximum expected reward, and that satisfies the given LTL property with maximal probability. We prove that maximising an expected reward is equivalent to

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

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

maximising the probability of satisfying the assigned LTL property. As mentioned above, in finite MDPs we also propose a mechanism to determine this probability while the agent is learning the MDP: consequently, we can certify the generated policy by quantifying how safe it is with respect to the LTL property. Furthermore, we show that whenever the probability of satisfying the given LTL property is zero, our algorithm is still able to produce "best available" policies.

In addition, we propose the concept of adaptive safe padding that forces the agent not to violate the LTL (or any general safety) property "during" the learning process. As it can be expected, the safe padding might limit the exploration in some safety-critical cases. However, we show that the proposed architecture is able to automatically handle the trade-off between efficient progress in exploration and ensuring strict safety. Theoretical guarantees are available on the convergence of the proposed algorithms. We also develop the required assumptions and theories, along with a formal reward shaping scheme for the satisfaction of the LTL property with maximum probability.

In this dissertation, we show the following hypotheses hold:

- **Model-free RL can be employed for LTL policy synthesis and probabilistic certification in finite MDPs under a provably-correct discounted reward shaping.**
- **LTL policy synthesis can be achieved efficiently using model-free (deep) RL in continuous-state continuous-action MDPs. The algorithm efficiency is shown in terms of training time, sample complexity, and success rate.**

In a nutshell, the proposed architecture performs efficiently and is compatible with RL algorithms that are at the core of recent developments in the community,

e.g. [15, 55]. Thus, the proposed approach, which we call **Logically-Constrained Reinforcement Learning (LCRL)**, might open up to further research in the area.

1.2 Related Work

This section reviews related work and discusses the assumptions that are made in other works to solve the problem of policy synthesis under LTL. An overview of these assumptions is given in Fig. 1.1, with a detailed discussion on these assumptions and their shortcomings in the rest of this section.

1.2.1 Full LTL

The problem of control policy synthesis in finite-state finite-action MDPs with temporal logic has been considered in numerous works. In [101], the property of interest is expressed in LTL, which is converted to a DRA using standard methods. A product MDP is then constructed with the resulting DRA and a modified DP is applied to the product MDP, maximising the worst-case probability of satisfying the specification over all transition probabilities. However, [101] assumes that the MDP is known a-priori. This assumption is relaxed in [104] and transition probabilities in the given MDP model are considered to be unknown. Instead, a Probably Approximately Correct MDP (PAC MDP) is constructed and further multiplied by the logical property after conversion to a DRA. The overall goal is to calculate a finite-horizon T-step value function for each state such that the obtained value is within an error bound from the probability of satisfying the given LTL property.

The problem of policy generation by maximising the probability of satisfying given unbounded reachability properties is investigated in [105]. The policy genera-

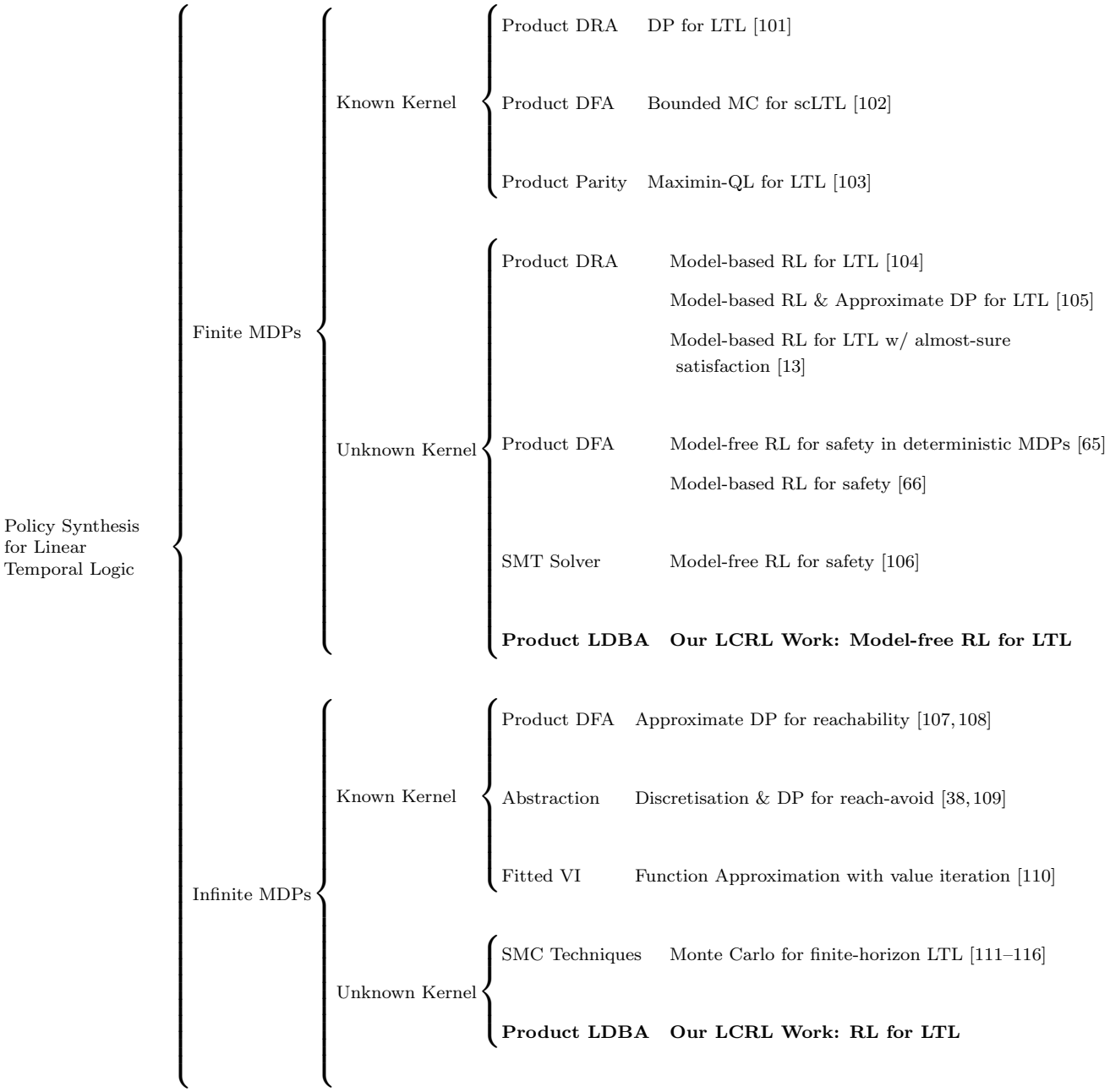


Figure 1.1: Related work overview.

tion relies on an approximate DP, even when the MDP transition probabilities are unknown. This requires a mechanism to approximate these probabilities (much like PAC MDP above), and the quality of the generated policy critically depends on the accuracy of this approximation. Therefore, a sufficiently large number of simulations has to be executed to make sure that the probability approximations are accurate enough [105]. Furthermore, the algorithm in [105] assumes prior knowledge about the smallest transition probability in the MDP. Using an LTL-to-DRA conversion, the algorithm given in [105] can be extended to the problem of control synthesis for LTL specifications, at the expense of double exponential blow-up of the obtained automaton. Much in the same direction, if there exists a policy whose traces satisfy the property with probability one [13] employs a learning-based approach to generate one such policy. For this approach, as remarked before, LTL-to-DRA conversion is in general known to result in large automata, and the reward shaping is complicated, due to the accepting conditions of the DRA. As for [105], the algorithm in [13] hinges on approximating the transition probabilities, which limits the precision and most importantly the scalability of the policy synthesis.

Compared to the mentioned approaches, our proposed scheme learns the dynamics of the MDP implicitly, whilst synthesising the optimal policy at the same time, hence without explicitly having to construct the transition probabilities or the MDP model first. Indeed, the proposed method can be implemented completely "model-free", which means that we are able to synthesise policies (1) without knowing the MDP graph and its transition probabilities (as opposed to DP); and (2) without preprocessing or constructing a model of the MDP (which is the base for, among other techniques, model-based RL). The second feature results in the synthesis of policies by direct interaction with the MDP. Moreover, unlike [13], the proposed algorithms are able to find the optimal policy even if the satisfaction probability is not equal to one. In addition to this, we show that whenever the probability of

satisfying the given LTL property is zero, our method is still able to produce "best available" policies.

Our work on model-free RL for LTL has been first introduced in [10] in 2018. Since then, growing research has been devoted to model-free RL with different kinds of automata [117], including limit-deterministic Büchi automata [118, 119], where different forms of reward schemes have been examined [120, 121]. Specifically, [118] employs a limit-deterministic Büchi automata based on a Non-deterministic Büchi Automaton (NBA) that is structurally different than [97]'s LDGBA. [119] extends upon the NBA-based automata, by proposing an interleaving reward and discounting scheme, and [7, 122] employs policy gradient to tackle MDPs with high-dimensional action and state spaces.

1.2.2 Sub-fragments of LTL

There has been significant interest also in specifying tasks in RL via sub-fragments of LTL [123–126] and other forms of temporal logic [127, 128]. The problem of synthesising a policy that satisfies a temporal logic specification and that at the same time optimises a performance criterion is considered in [103, 129–131]. In [102], scLTL is proposed for mission specifications, which results in Deterministic Finite Automatons (DFAs). A product MDP is then constructed and a linear programming solver is used to find optimal policies. Algorithms proposed in [5, 25] synthesise DFAs on-the-fly in the context of deep RL and inverse RL, to infer an scLTL property. Conversely, Mealy machines are inferred in [132] to represent a temporal non-Markovian task in Monte Carlo tree search. PCTL specifications are investigated in [133], where a linear optimisation solution is used to synthesise a control policy. In [134], an automated method is proposed to verify and repair the policies that are generated by RL with respect to a PCTL formula - the key engine runs by feeding the Markov chain induced by the policy to a probabilistic

model checker. In [135], some practical challenges of RL are addressed by letting the agent plan ahead in real-time using constrained optimisation.

In [103], the problem is separated 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. Similarly, [106] first computes safe, permissive strategies with respect to a reachability property. Then, under these constrained strategies, RL is applied to synthesise a policy that satisfies an expected cost criterion. The concept of shielding is employed in [65] to synthesie a policy that ensures that the agent remains safe during and after learning for a fully deterministic reactive system. This approach is closely related to teacher-guided RL [136], since a shield can be considered as a teacher, which provides safe actions when absolutely necessary. To express the temporal specification, [65] uses DFAs and then translates the problem into a safety game. The game is played between the environment and the agent, where in every state of the game, the environment chooses an input, and then the agent selects an output. The game is won by the agent if only safe states are visited during the play. However, the generated policy always needs the shield to be online, as the shield maps every unsafe action to a safe action.

The work in [66] extends [65] to probabilistic systems modelled as MDPs with adversarial uncontrollable agents. The general assumption in [66] is that the controllable agent acquires full observation over the MDP and adversarial agents: unlike the proposed scheme in this dissertation, the RL scheme used in [66] is model-based and requires the agent to first build a model of the stochastic MDP. The concept of bounded-prescience shield is proposed in [6] for analysing and ensuring the safety of deep RL agents in Atari games. [137–140] address safety-critical settings in the context of cyber-physical systems, where the agent has to

deal with a heterogeneous set of models in model-based RL. [138] first generates a set of feasible models given an initial model and data on runs of the system. With such a set of feasible models, the agent has to learn how to safely identify which model is the most accurate one. [141] further employs differential dynamic logic [142], a first-order multimodal logic for specifying and proving properties of hybrid models.

Safe RL is an active area of research whose focus is on the efficient implementation of safety properties, and is mostly relied on reward engineering [70]. Our proposed method is related to safe RL, but cannot simply be reduced to it, due to its generality and to its inherent structural differences. By focusing on the safety fragment of LTL, LCRL does not require the reward function to be handcrafted. The reward function is automatically shaped by exploiting the structure of the LDBA and its generalised Büchi acceptance condition. However, for the safety fragment of LTL, the proposed automatic reward shaping can be seen as a way of "modifying the optimization criterion," as in [70].

Additionally, we would like to emphasise that our work cannot be considered a Constrained MDP (CMDP) method, as the LTL satisfaction is encoded in the expected return itself, while in CMDP algorithms the original objective is separated from the constraint. In a nutshell, LCRL inherits reward engineering aspects that are standard in safe RL, however, at the same time, it infuses notions from formal methods that allow guiding exploration and certifying its outcomes. Note that safe RL in general is not equivalent to ensuring agent safety during learning. Learning-while-being-safe by itself is a semantically-different research area that has attracted considerable attention recently, e.g. [3, 143–149]

1.2.3 LTL for Uncountably Infinite State or Action Spaces

At the time of writing this dissertation in early 2020, in the domain of continuous-state continuous-action MDPs, no research had been done to enable RL to generate policies that satisfy full LTL properties. The scheme discussed in this work, i.e. [1] published in 2019 and [4] published in 2020, were the first algorithms that could handle both finite-state and continuous-state MDPs in this context. Fortunately, this topic has attracted researchers attention and different approaches have been proposed to tackle this problem [122, 126]. More work has been done if the model of the MDP is known, as detailed next. Probabilistic reachability over a finite horizon for hybrid continuous-state MDPs is investigated in [107], where a DP-based algorithm is employed to produce safe policies. DFAs have been employed in [108] to find an optimal policy for infinite-horizon probabilistic reachability problems. FAUST^2 [38] deals with uncountable-state MDPs by generating a discrete-state abstraction based on the knowledge of the MDP model. Using probabilistic bi-simulation, [150] showed that abstraction-based model checking can be effectively employed to generate control policies in continuous-state MDPs. Bounded LTL is proposed in [151] as the specification language, and a policy search method is used for synthesis.

Statistical Model Checking (SMC) techniques have also been studied for policy synthesis in MDPs, however, they are not well suited to models that exhibit non-determinism. This is due to the fact that SMC techniques often rely on the generation of random paths, which are not well-defined for an MDP with non-determinism [111, 112]. Some SMC approaches proposed to resolve the MDP non-determinism by using uniform distributions [113, 114] and others proposed to consider all possible strategies [115, 116] and produced policies that are close to an optimal one. Unlike RL, which improves its exploration policy during learning, a

constant random policy is expected to waste time and computational resources to generate sample traces. Also, a trace is "only" used to reinforce each state-action pair visited by the associated path if the trace satisfies the property of interest. This is quite similar to Monte Carlo methods rather than RL or DP. For this reason, SMC methods are not expected to scale as well as RL. Further, sampling and checking of traces need to be computationally feasible: SMC techniques are effective with finite-horizon LTL properties, as opposed to the focus of this work on infinite-horizon properties and full LTL. The efforts on statistical model-checking of unbounded properties are limited to a few specifications [152]. However, there have been some recent developments, e.g. [116], that leverage RL to reduce the randomness in the policy that resolves the MDP non-determinism. Despite significant improvements, SMC techniques are still limited to finite-horizon LTL and Monte Carlo search.

1.2.4 Summary

We examined previous works on policy synthesis for LTL and its sub-fragments. As discussed, model-based RL approaches in LTL synthesis do not scale to problems with large state and action spaces. This issue is resonated when the product MDP is explicitly constructed with the learned model and DRA. Further, we reviewed that in general LDBAs are much smaller and easier to implement than DRAs in RL.

This section also discussed LTL policy synthesis in continuous-state continuous-action MDPs and that no RL algorithm had been proposed prior to our work to solve this problem. The classical approaches available to tackle this problem are either discretisation-based or SMC-based, where each having critical drawbacks such as loss of dynamics accuracy or convergence issues.

1.3 Novelty Claims

We propose Logically-Constrained Reinforcement Learning (LCRL), a new architecture for LTL synthesis and quantitative model checking. In short, the novel aspects of LCRL are as follows:

1. **The first model-free RL method for LTL policy synthesis**

   LCRL is the first model-free RL technique that can synthesise policies whose traces satisfy a given LTL property with maximum probability. Specifically, we show that if suitable assumptions hold, under the proposed reward shaping, LCRL policy maximises the property satisfaction probability. Our LCRL paper has been available on arXiv since January 24, 2018 [10]. All previous works on RL for full LTL policy synthesis in finite-state-action MDPs are purely model-based [13, 104, 105].

2. **Enhancement of scalability and convergence rate LTL policy synthesis**

   We observed an improvement of one order of magnitude in the number of iterations required for the policy synthesis compared to model-based approaches with DRA [13, 104, 105] in finite MDPs and [38, 109, 110, 153] in infinite MDPs. Without a need to build a model of the MDP and without constructing the product MDP, LCRL is able to minimise the sample complexity of LTL policy synthesis comparing to model-based approaches. This essentially allows LCRL to converge faster, and be applied to MDPs with larger state and action spaces.

3. **The first RL synthesis algorithm for LTL in continuous-state-action MDPs**

   In the domain of continuous-state and also the more general case of continuous-

state continuous-action MDPs, no research had been done to enable RL, model-based or model-free, to synthesise policies for LTL prior to LCRL. LCRL is the first method in this area, against which we compared the most well-known methods to deal with continuous spaces [38, 107, 109, 110, 154] (Fig. 1.1). More specifically, [38, 107, 109, 154] automatically discretise continuous-state MDPs to a finite-state MDP, over which we vanilla RL can be run. Alternatively, [110] proposes a DP-based method with function approximation for the value function.

4. **Safe padding to efficiently ensure the agent safety during learning with minimum number of extra assumptions**

   Safe exploration is the key enabling element of applications of RL in safety-critical systems. For instance, robotic machines with certain specified tasks, working in proximity of humans, need to be able to learn the assigned task optimally while never causing an injury or damage. LCRL presents the concept of adaptive safe padding that forces an RL agent to synthesise optimal control policies while ensuring safety during the learning process. Recent approaches to safe exploration [65, 66, 155–160] are either computationally expensive or require explicit, strong assumptions about the model of agent-environment interactions. By contrast, the safe padding approach is computationally efficient and relaxes a-priory assumptions on the underlying MDP.

5. **Automatic task decomposition and one-shot learning in hierarchical RL**

   The reward function proposed in this work is automatically shaped on-the-fly with no supervision, allowing one to also automatically modularise the global complex task into easy sub-tasks. Hierarchical RL methods often require detailed supervision in the form of explicitly specified high-level actions or

intermediate supervisory signals [87–92]. Another closely-related work is curriculum learning [92], which masters easy instruction-based tasks first and eventually composes them together, to accomplish a more complex task. In LCRL on the other hand, the complex task is expressed as an LTL instruction to guide learning and to directly generate policies, with no need to start from easier tasks and to later join corresponding policies together. This allows LCRL to learn sub-tasks simultaneously in a one-shot[3] learning scenario.

1.4 Dissertation Overview

We propose a general scheme to guide an RL agent by expanding the agent domain knowledge about the environment by means of an LTL property. This additional knowledge boosts the performance of the agent to learn a globally optimal policy for the specified property. Further, we show that we can calculate the probability that is associated with the satisfaction of the LTL property that enables us to quantify the safety of the generated optimal policy at any given state.

This work is the first RL for LTL research that converts the LTL property to a GBA-based LDBA that consequently results in a significantly smaller automaton than DRA alternatives, and increases the convergence rate of RL. 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 due to the simpler accepting conditions of the LDBA.

Much like the way we synchronised the states of the LDBA with the states of the MDP, we have shown that synchronising a Kripke structure with the LDBA allows us to handle time-varying periodic environments on-the-fly. This particularly becomes important when we employed this synchronised LDBA-Kripke automaton

[3]One-shot learning in the RL context means that the learning algorithm is able to synthesise policies from relatively few experience samples, and instantly generalize to new situations of the same task, without requiring task-specific engineering.

as an infrastructure for the agent to transfer its learning over the dimension of time and to overcome the curse of dimensionality.

Last but not least, this work presents the first (deep) neural-network-based RL method that achieved LTL policy synthesis in a continuous-state continuous-action MDP. Our algorithm is model-free, meaning that the learning only depends on the sample experiences that the agent gathered by interacting and exploring the MDP. Further, the sample set can be small thanks to the generalisation that deep neural nets offer. The core engine of our algorithm is very flexible and can be extended to the most recent developments in RL literature. In particular, in order to show the enhancement of learning within the proposed architecture, we have picked the Atari 2600 game "Montezuma's Revenge" as one of our case studies, which is the only game in [15] that DQN fails to gain any score at.

A breakdown of the structure of this dissertation is as follows:

- **Chapter 2** introduces fundamental notation and concepts in RL. This is followed by defining the syntax and semantics of LTL, which becomes the basis of the proposed algorithms in the remaining chapters. Next, we formally provide a definition for LDBA, the automaton we proposed to use as the state of the art in order to express the given LTL formula.

- **Chapter 3** discusses our proposed general architecture Logically-Constrained Reinforcement Learning (LCRL), and the associated notions that are central to this scheme. LCRL is the first model-free RL method that synthesises policies for LTL. In later chapters, we show how LCRL is employed to tackle MDPs with finite and also uncountably infinite state and action spaces.

- **Chapter 4** investigates the capability of LCRL when applied to finite-state finite-action MDPs. Owing to the underlying structure of the LDBA, LCRL is able to guide the exploration so as to minimize the solution time by only

considering the portion of the MDP that is relevant to the LTL property. This is, in particular, made clear in a Pacman experiment, where we show that classical RL is very slow and ineffective in finding the optimal policy, whereas LCRL converges extremely fast thanks to the guidance that is provided by the LDBA. Additionally, in this chapter, we propose an off-policy asynchronous value iteration method to calculate the probability of satisfaction of the LTL property along with RL. We show that we are able to extend the satisfaction probability calculation to large-scale MDPs, which are hard for DP-based methods to handle.

- **Chapter 5** proposes Logically-Constrained Neural Fitted Q-iteration (LCNFQ), the first RL algorithm to train a Q-function in a continuous-state MDP such that the resulting traces satisfy an LTL property. LCNFQ exploits the positive effects of generalization in neural networks while at the same time avoid the negative effects of disturbing previously learned experiences. We show that LCNFQ requires less experience and the learning process is highly data-efficient which subsequently increases the scalability and applicability of the proposed algorithm. LCNFQ is model-free, meaning that the learning only depends on the sample experiences that the agent gathered by interacting and exploring the MDP. This chapter shows that LCNFQ is successfully tested in numerical examples to verify its performance and it outperformed the competitors.

- **Chapter 6** pushes the scalability limits of LCRL and proposes an actor-critic, model-free, and online deep RL scheme for continuous-state continuous-action MDPs when the reward is highly sparse. The LDBA acts as a guide for the agent within the product, where a modular Deep Deterministic Policy Gradient (DDPG) architecture is proposed to generate a low-level

control policy. This synchronisation lifts deep RL applicability to complex temporal, non-Markovian, and memory-dependent policy synthesis problems. We evaluate our algorithm in two Mars rover experiments and we achieve near-perfect success rates while standard DDPG fails.

- **Chapter 7** introduces Cautious Reinforcement Learning, a general method for safe exploration in RL, 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. However, in this chapter, we extend the LCRL idea to employ LTL to express an overall task (or goal), and to shape the reward for the agent in a provably-correct and safe scheme. We propose 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, learns a policy that satisfies the LTL requirement. The pessimistic agent creates a continuously growing "safe padding" for the optimistic agent, which can learn an 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.

- **Chapter 8** concludes the dissertation by reviewing the main contributions of this work and outlines directions for future research.

1.5 Publications by the author

The material presented in this dissertation has appeared or is under review in international conference proceedings or peer-reviewed journals. The connection between each chapter and the publications is as follows:

- **Chapter 3 and 4**

  Hasanbeig, M., Kantaros, Y., Abate, A., Kroening, D., Pappas, G. J., and Lee, I., "Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees", IEEE Conference on Decision and Control (CDC), 2019. [2]

  – The first two authors contributed equally in idea development and manuscript writing. Algorithm implementation and proofs are mostly done by the first author. The first two authors received valuable guidance from the co-authors.

  – The materials presented in Sections 3.2, 3.3, (idea development) and 4.3 (algorithm implementation) have been published in [2]. Section 4.2 (proofs), excluding Subsections 4.2.1 and 4.2.2, is also presented in [2].

  Hasanbeig, M., Abate, A. and Kroening, D., "Certified Reinforcement Learning with Logic Guidance", CoRR abs/1902.00778, 2019. [12]

  – The first author was responsible for idea development, algorithm implementation, manuscript writing and proofs, and he received valuable guidance from the co-authors.

  – The materials presented in Sections 3.2, 3.3, (idea development), 4.2 (proofs and deep analysis) and 4.3 (algorithm implementation) have been published in [12].

  Hasanbeig, M., Abate, A. and Kroening, D., "Logically-Constrained Reinforcement Learning", CoRR abs/1801.08099, 2018. [10]

  – The first author was responsible for idea development, algorithm implementation, manuscript writing and proofs, and he received valuable guidance from the co-authors.

  – The materials presented in Sections 3.2, 3.3, (idea development), and some parts of 4.2 (proofs) have been published in [10]. More specifically, from

Section 4.2 Subsection 4.2.1 is the focus of [10].

- **Chapter 5**

  Hasanbeig, M., Abate, A. and Kroening, D., "Logically-Constrained Neural Fitted Q-Iteration", International Conference on Autonomous Agents and Multi-agent Systems (AAMAS), 2019. [1]

  – The first author was responsible for idea development, algorithm implementation, manuscript writing and proofs, and he received valuable guidance from the co-authors.

  – The materials presented in Chapter 5 have been published in [1].

- **Chapter 6**

  Hasanbeig, M., Kroening, D. and Abate, A., "Deep Reinforcement Learning with Temporal Logics", International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), 2020. [4]

  – The first author was responsible for idea development, manuscript writing and proofs, and he received valuable guidance from the co-authors.

  Cai, M., Hasanbeig, M., Xiao, S., Abate, A., and Kan, Z., "Modular Deep Reinforcement Learning for Continuous Motion Planning with Temporal Logic", International Conference on Intelligent Robots and Systems (IROS), 2021. [7]

  – The first two authors contributed equally in manuscript writing. The first two authors received valuable guidance from the co-authors.

  – The materials presented in Chapter 6 have been published in [4, 7].

- **Chapter 7**

  Hasanbeig, M., Abate, A. and Kroening, D., "Cautious Reinforcement Learning with Logical Constraints", International Con-

ference on Autonomous Agents and Multi-agent Systems (AAMAS), 2020. [3]

– The first author was responsible for idea development, algorithm implementation, manuscript writing and proofs, and he received valuable guidance from the co-authors.

– The materials presented in Chapter 7 have been published in [3].

Hasanbeig, M., Kroening, D. and Abate, A., "Towards Verifiable and Safe Model-Free Reinforcement Learning", Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY), 2020. [11]

– The first author was responsible for idea development, algorithm implementation, manuscript writing and proofs, and he received valuable guidance from the co-authors.

– Publication [11] is a high-level overview of the materials presented in Chapter 7.

The following publications [5, 6] are closely related to the material presented in later chapters, but are not fully included within this dissertation:

Hasanbeig, M., Jeppu, N. Y., Abate, A., Melham, T., and Kroening, D., "DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning", AAAI Conference on Artificial Intelligence (AAAI), 2021.

The above proposes DeepSynth, a method for effective training of deep Reinforcement Learning (RL) agents when the reward is sparse and non-Markovian, but at the same time progress towards the reward requires achieving an unknown sequence of high-level objectives. Our method employs a novel algorithm for

synthesis of compact automata to uncover this sequential structure automatically. We synthesise a human-interpretable automaton from trace data collected by exploring the environment. The state space of the environment is then enriched with the synthesised automaton so that the generation of a control policy by deep RL is guided by the discovered structure encoded in the automaton. The proposed approach is able to cope with both high-dimensional, low-level features and unknown sparse non-Markovian rewards. We have evaluated DeepSynth's performance in a set of experiments that includes the Atari game "Montezuma's Revenge". Compared to existing approaches, we obtain a reduction of two orders of magnitude in the number of iterations required for policy synthesis, and also a significant improvement in scalability.

Giacobbe, M., Hasanbeig, M., Kroening, D., and Wijk, H., "Shielding Atari Games with Bounded Prescience", International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), 2021.

Deep reinforcement learning (DRL) is applied in safety-critical domains such as robotics and autonomous driving. It achieves superhuman abilities in many tasks, however, whether DRL agents can be shown to act safely is an open problem. Atari games are a simple yet challenging exemplar for evaluating the safety of DRL agents and feature a diverse portfolio of game mechanics. The safety of neural agents has been studied before using methods that either require a model of the system dynamics or an abstraction; unfortunately, these are unsuitable to Atari games because their low-level dynamics are complex and hidden inside their emulators. The above work presents the first exact method for analysing and ensuring the safety of DRL agents for Atari games. Our method only requires access to the emulator. First, we give a set of properties that characterise "safe behaviour" for several games. Second, we develop a method for exploring all traces

induced by an agent and a game and consider a variety of sources of game non-determinism. We observe that the best available DRL agents reliably satisfy only very few properties; several critical properties are violated by all agents. Finally, we propose a countermeasure that combines a bounded explicit state exploration with shielding. We demonstrate that our method improves the safety of all agents over multiple properties.

Lim Zun Yuan, Hasanbeig, M., Abate, A. and Kroening, D., "Modular Deep Reinforcement Learning with Temporal Logic Specifications", CoRR abs/1909.11591, 2019.

In this paper we propose an actor-critic, model-free, and online Reinforcement Learning (RL) framework for continuous state continuous-action Markov Decision Processes (MDPs) when the reward is highly sparse but encompasses a high-level temporal structure. We represent this temporal structure by a finite-state machine and construct an on-the-fly synchronised product with the MDP and the finite machine. The temporal structure acts as a guide for the RL agent within the product, where a modular Deep Deterministic Policy Gradient (DDPG) architecture is proposed to generate a low-level control policy. We evaluate our framework in a Mars rover experiment and we present the success rate of the synthesised policy

Chapter 2

Background

This chapter introduces notation and well-known concepts in RL together with the syntax and semantics of LTL. We establish the underpinnings of the algorithms proposed in the remaining chapters as we review the fundamentals of RL and the basics of the associated algorithms. Next, we formally introduce LTL and we provide the definition of the LDBA that we used in this work.

2.1 Preliminaries

Definition 2.1 (General MDP [161, 162]). *The tuple* $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ *is a general MDP over a set of continuous states* $\mathcal{S} = \mathbb{R}^n$*, where* $\mathcal{A} = \mathbb{R}^m$ *is a set of continuous actions, and* $s_0 \in \mathcal{S}$ *is the MDP initial state.* $P : \mathcal{B}(\mathcal{S}) \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ *is a Borel-measurable conditional transition kernel which assigns to any pair of state* $s \in \mathcal{S}$ *and action* $a \in \mathcal{A}$ *a probability measure* $P(\cdot|s, a)$ *on the Borel space* $(\mathcal{S}, \mathcal{B}(\mathcal{S}))$*, where* $\mathcal{B}(\mathcal{S})$ *is the set of all Borel sets on* $\mathcal{S}$*.* $\mathcal{AP}$ *is a finite set of atomic propositions and a labelling function* $L : \mathcal{S} \to 2^{\mathcal{AP}}$ *assigns to each state* $s \in \mathcal{S}$ *a set*

of atomic propositions $L(s) \subseteq 2^{\mathcal{AP}}$.

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

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

In this work, the general assumption is that the MDP $\mathfrak{M}$ is fully unknown, and the learning agent has no prior knowledge about the transition kernel P and the labelling function L.

Definition 2.2 (**Stationary and 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. A stationary deterministic policy is then a mapping from states to actions* $\pi : \mathcal{S} \to \mathcal{A}$.

Definition 2.3 (**Expected Infinite-Horizon Discounted Return [14]**). *For any policy* π *on an MDP* $\mathfrak{M}$*, and given a reward function* R*, the expected discounted reward at state* s *is defined as:*

$$\begin{aligned} U^{\pi}_{\mathfrak{M}}(s) &= \mathbb{E}^{\pi}[\sum_{n=0}^{\infty} \gamma^n\ r_n | s_0 = s], \\ s_n &\sim P(\cdot|s_{n-1}, a_{n-1}),\ a_n \sim \pi(\cdot, s_n) \end{aligned} \tag{2.1}$$

where $\mathbb{E}^{\pi}[\cdot]$ *denotes the expected value given that the agent follows policy* π *from*

state s, $\gamma \in [0,1)$ ($\gamma \in [0,1]$ when episodic) is a discount factor, and $s_0, a_0, s_1, a_1 ...$ is the sequence of states generated by policy π, initialised at $s_0 = s$. We drop the subscript $\mathfrak{M}$ when it is clear from the context.

Note that the discount factor γ is a hyper-parameter that has to be tuned. In particular, there is standard work in RL on state-dependent discount factors [163–166], which is shown to preserve convergence and optimality guarantees. A possible tuning strategy for our setup would let the discount factor be a state-dependent function

$$\gamma(s) = \begin{cases} \eta & \text{if } R(s,a) > 0, \\ 1 & \text{otherwise,} \end{cases} \tag{2.2}$$

where $0 < \eta < 1$ is a constant. Hence, (2.1) would reduce to the following [166]:

$$U^{\pi}_{\mathfrak{M}}(s) = \mathbb{E}^{\pi}[\sum_{n=0}^{\infty} \gamma(s_n)^{N(s_n)}\ r_n | s_0 = s],\ 0 \leq \gamma \leq 1, \tag{2.3}$$

where $N(s_n)$ is the number of times a positive reward has been observed at state s_n. Of course, there might be alternatives to this tuning strategy: we do not pursue this direction as it is not central to our problem setup.

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

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

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

Theorem 2.1 ([29, 167]). *In an MDP $\mathfrak{M}$ with a bounded reward function and a finite action space optimal policies are stationary and deterministic.*

By Theorem 2.1, as long as the reward function is bounded and the action space is finite, the optimal policy in Definition 2.4 exists.

An MDP $\mathfrak{M}$ is said to be solved if the agent discovers an optimal policy π^* that maximises the expected return. In the following, we provide the necessary background on model-free RL algorithms that we use in this work.

2.2 Background on Reinforcement Learning

This section separates the presentation for finite-state-action and the infinite-state-action MDPs. We emphasise that the infinite-state-action algorithm, which includes a generalisation step, can also handle the problem of policy synthesis in the finite case and can in particular be useful when the cardinality of the state space or action space is very large. However, as we show later if quantitative certificates are required in the finite-state case, we need to resort to the specific results that are provided for finite-state MDPs.

2.2.1 Q-learning

Q-learning (QL) is the most extensively used RL algorithm for synthesising optimal policies in finite-state MDPs [14]. For each state $s \in \mathcal{S}$ and for any available action $a \in \mathcal{A}$, QL assigns a quantitative value $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which is initialised with an arbitrary and finite value over all state-action pairs. As the agent starts receiving rewards during learning, the Q-function is updated by the following rule for taking action a at state s:

$$Q(s,a) \leftarrow Q(s,a) + \mu[R(s,a) + \gamma \max_{a' \in \mathcal{A}}(Q(s',a')) - Q(s,a)], \tag{2.4}$$

where $Q(s,a)$ is the Q-value corresponding to state-action (s,a), $0 < \mu \leq 1$ is called learning rate or step size, $R(s,a)$ is the reward obtained for performing action a in state s, $0 \leq \gamma \leq 1$ is the discount factor, and s' 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^*, as long as every state-action pair is visited infinitely often [33]. This unique limit is the expected discounted return by taking action a at state s, and following the optimal policy afterwards. Once QL converges, the optimal policy π^* can be generated by selecting the action that yields the highest Q^*, i.e.,

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \; Q^*(s, a).$$

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

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') U^{\pi^*}(s'), \tag{2.5}$$

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

2.2.2 Neural Fitted Q-iteration

In QL the agent stores the Q-values over all state-action pairs in a look-up table, and updates them according to rule (2.4). Evidently, when the MDP state space is uncountably infinite it is not possible to directly use the standard QL look-up table. Thus, we have to either finitely discretise the state space or to turn to function approximations, in order to interpolate the Q-function over the entire uncountably infinite state space.

Neural Fitted Q-iteration (NFQ) [56] is an algorithm that employs feedforward neural networks [168] to approximate the Q-function, namely to efficiently generalise or interpolate it over the entire state space, exploiting a finite set of experience

samples. NFQ is also the core engine behind the well-known DQN [15] architecture.

Instead of the update rule (2.4), NFQ introduces a loss function that measures the error between the current Q-values $Q(s, a)$ and their target value $R(s, a) + \gamma \max_{a'} Q(s', a')$, namely

$$L = [Q(s, a) - R(s, a) + \gamma \max_{a'} Q(s', a')]^2. \tag{2.6}$$

Gradient descent techniques [169] are then applied to adjust the weights of the Q-function neural network, so that this loss is minimised.

In classical QL, the Q-function is updated whenever a state-action pair is visited. In the continuous state-space case, we may update the approximation likewise, i.e., update the neural net weights once a new state-action pair is visited. However, in practice, a large number of training iterations might need to be carried out until an optimal or near-optimal policy is found. This is due to the uncontrollable variations occurring in the Q-function approximation caused by unpredictable changes in the network weights when the weights are adjusted for a specific state-action pair [170].

More precisely, if at each iteration we only introduce a single sample point the training algorithm tries to adjust the weights of the entire neural network, such that the loss function is minimised at that specific sample point. This might result in changes in the network weights such that the difference between the network output and the output for previous sample points becomes large and thus the algorithm fails to approximate the Q-function correctly. Therefore, one needs to ensure that when the weights of the neural network are updated, all the previously generated samples are considered: this technique is called "experience replay" [171], and is detailed later. The main idea underlying NFQ is to store all previous experiences and then reuse this data iteratively to update the neural Q-function. NFQ can thus be seen as a batch learning method in which there exists a training set that is repeatedly used to train the agent. In other words, experience gathering and

learning happen separately.

2.2.3 Deep Deterministic Policy Gradient

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

Policy gradient methods, on the other hand, are online schemes that are widely used in RL for MDPs with continuous action spaces. The general idea is to represent the policy by a parametric probability distribution $\pi(\cdot|s, \theta^\pi)$ and then adjusting the policy parameters θ^π in the direction of the greatest cumulative reward. This policy can of course be deterministic, but there is a crucial difference between the stochastic and deterministic policy gradients [172]. From a practical point of view, stochastic policy gradient requires more experience samples. In this work, we focus on deterministic policies as they are sufficient for most control problems and because deterministic policy gradients are more efficient in terms of sample complexity.

The actor-critic architecture is a widely used method based on the policy gradient [14], which consists of two interacting components: an actor and a critic. The actor is the parametric policy $\pi(\cdot|s, \theta^\pi)$ and the critic is an action-value function

$Q(s, a)$ that guides the updates of parameters θ^π in the direction of the greatest cumulative reward. The Deterministic Policy Gradient (DPG) algorithm [173] introduces a parameterised deterministic function $\pi(s|\theta^\pi)$ as the actor to represent the current policy by deterministically mapping states to actions, where θ^π is the function approximation parameters for the actor function. A parameterised action-value function $Q(s, a|\theta^Q)$ is the critic and is learned as described next.

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

$$L(\theta^Q) = \mathbb{E}^\pi_{s_n \sim \rho^\beta}[(Q(s_n, a_n|\theta^Q) - y_n)^2], \tag{2.7}$$

where ρ^β is the probability distribution of state visits over $\mathcal{S}$ under any given arbitrary stochastic policy β, and $y_n = R(s_n, a_n) + \gamma Q(s_{n+1}, a_{n+1}|\theta^Q)$ such that $a_{n+1} = \pi(s_{n+1})$.

The parameters of the actor $\pi(s|\theta^\pi)$ are updated by applying the chain rule to the expected return with respect to the actor parameters, which is approximated as follows [173]:

$$\begin{aligned} &\nabla_{\theta^\pi} U^\pi(s_n) \approx \mathbb{E}_{s_n \sim p^\beta}[\nabla_{\theta^\pi} Q(s, a|\theta^Q)|_{s=s_n, a=\pi(s_n|\theta^\pi)}] \\ &= \mathbb{E}_{s_n \sim p^\beta}[\nabla_a Q(s, a|\theta^Q)|_{s=s_n, a=\pi(s_n)} \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s=s_n}]. \end{aligned} \tag{2.8}$$

The results in [173] show that this is a policy gradient, and therefore we can apply a policy gradient algorithm on the deterministic policy. Deep DPG (DDPG) further extends DPG by employing a deep neural network as a function approximator and updating the network parameters via a "soft update" method, similar to [15] and is thoroughly explained later.

So far we discussed the fundamentals of a number of well-known RL algorithms that seen expanded use and increased success. As proposed before, in this dissertation we would like to leverage the success of RL in learning high-performance controllers while guaranteeing the learning outcomes. To do so, we employ LTL and LDBA, which we formally define in the next section as interpreted over MDPs.

2.3 Linear Temporal Logic

In the proposed architecture, we use LTL [83] formulae to express a wide range of properties (e.g., temporal, sequential, conditional), and to systematically and automatically shape a corresponding reward: such a reward would otherwise be hard (if at all possible) to express and achieve by conventional handcrafted methods in classical reward shaping.

Definition 2.5 (**Path**). *In an MDP $\mathfrak{M}$, an infinite path ρ starting at s_0 is a sequence of states $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ such that every transition $s_i \xrightarrow{a_i} s_{i+1}$ is possible in $\mathfrak{M}$, i.e. s_{i+1} belongs to the smallest Borel set B such that $P(B|s_i, a_i) = 1$ (or in a finite-state MDP, s_{i+1} is such that $P(s_{i+1}|s_i, a_i) > 0$). We might also denote ρ as $s_0..$ to emphasize that ρ starts from s_0.*

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

LTL formulae over a given set of atomic propositions $\mathcal{AP}$ are syntactically defined as [83]

$$\varphi ::= \mathit{true} \mid \alpha \in \mathcal{AP} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi \ \mathrm{U} \ \varphi, \tag{2.9}$$

where the operators $\bigcirc$ and U are called "next" and "until", respectively. The

semantics of LTL formulae, as interpreted over MDPs, are discussed in the following.

Definition 2.6 (**LTL Semantics [83]**). *For an LTL formula φ and for a path ρ in an MDP $\mathfrak{M}$, the satisfaction relation $\rho \models \varphi$ is defined as*

$$\rho \models \alpha \in \mathcal{AP} \Leftrightarrow \alpha \in L(\rho[0]),$$

$$\rho \models \varphi_1 \wedge \varphi_2 \Leftrightarrow \rho \models \varphi_1 \wedge \rho \models \varphi_2,$$

$$\rho \models \neg\varphi \Leftrightarrow \rho \not\models \varphi,$$

$$\rho \models \bigcirc\varphi \Leftrightarrow \rho[1..] \models \varphi,$$

$$\rho \models \varphi_1 \mathrm{U} \varphi_2 \Leftrightarrow \exists j \in \mathbb{N}_0 \ s.t. \ \rho[j..] \models \varphi_2 \ \wedge \ \forall i, \ 0 \leq i < j, \ \rho[i..] \models \varphi_1.$$

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

$$Words(\varphi) = \{\sigma \in (2^{\mathcal{AP}})^{\omega} \text{ s.t. } \sigma \models \varphi\}. \tag{2.10}$$

Definition 2.7 (**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*

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

where $s..^{\pi}$ is the collection of all paths starting from s, generated under policy π.

The maximum probability of satisfaction is also defined as:

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

Definition 2.8 (**Policy Satisfaction**). *In an MDP $\mathfrak{M}$, we say that a stationary deterministic policy π satisfies an LTL formula φ if:*

$$Pr(s_0..^{\pi} \models \varphi) > 0,$$

where s_0 is the initial state of the MDP.

Using an LTL formula we can now specify a set of constraints (i.e., requirements, or specifications) over the traces of the MDP. Once a policy π is selected, it dictates which action has to be taken at each state of the MDP $\mathfrak{M}$. Hence, the MDP $\mathfrak{M}$ is reduced to a Markov chain, which we denote by $\mathfrak{M}^{\pi}$. The Markov chain $\mathfrak{M}^{\pi}$ will be later used to prove the optimality of a given policy π with respect to the LTL property.

For an LTL formula φ, an alternative method to express the set $Words(\varphi)$ in (2.10) is to employ a limit-deterministic Büchi automaton (LDBA) [97]. We first define a Generalised Büchi Automaton (GBA), then we formally introduce the LDBA [97].

Definition 2.9 (**Generalised Büchi Automaton**). *A GBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is a structure where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $\Sigma = 2^{\mathcal{AP}}$ is a finite alphabet, $\mathcal{F} = \{F_1, ..., F_f\}$ is the set of accepting conditions, where $F_j \subset \mathcal{Q}, 1 \leq j \leq f$, and $\Delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$ is a transition relation.*

Let Σ^{ω} be the set of all infinite words over Σ. An infinite word $\sigma \in \Sigma^{\omega}$ is accepted by a GBA $\mathfrak{A}$ if there exists an infinite run $\theta \in \mathcal{Q}^{\omega}$ starting from q_0 where

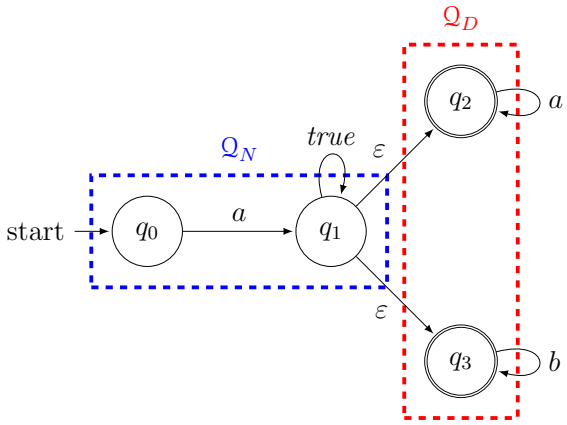


Figure 2.1: LDBA for the formula $a \wedge \bigcirc(\Diamond\Box a \vee \Diamond\Box b)$.

$\theta[i+1] \in \Delta(\theta[i], \sigma[i]),\ i \geq 0$ and for each $F_j \in \mathcal{F}$

$$inf(\theta) \cap F_j \neq \emptyset, \tag{2.11}$$

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

Definition 2.10 (LDBA [97]). *A GBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is limit-deterministic if $\mathcal{Q}$ can be partitioned into two disjoint sets $\mathcal{Q} = \mathcal{Q}_N \cup \mathcal{Q}_D$, such that:*

- *$\Delta(q, \alpha) \subset \mathcal{Q}_D$ and $|\Delta(q, \alpha)| = 1$ for every state $q \in \mathcal{Q}_D$ and for every $\alpha \in \Sigma$,*
- *for every $F_j \in \mathcal{F}$, $F_j \subset \mathcal{Q}_D$,*
- *$q_0 \in \mathcal{Q}_N$, and all the transitions from $\mathcal{Q}_N$ to $\mathcal{Q}_D$ are non-deterministic ε-transitions.*[1]

Intuitively, the defined LDBA is a GBA that has two partitions: initial ($\mathcal{Q}_N$) and accepting ($\mathcal{Q}_D$). The accepting part includes all the accepting states and has deterministic transitions. As an example, Fig. 2.1 shows the LDBA constructed for the formula $\varphi = a \wedge \bigcirc(\Diamond\Box a \vee \Diamond\Box b)$.

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

Remark 2.1. *The LTL-to-LDBA algorithm proposed in [97], which is used in this work, results in an automaton with two parts (initial $\mathcal{Q}_N$ and accepting $\mathcal{Q}_D$). Both initial and accepting parts comprise deterministic transitions, and additionally there are non-deterministic ε-transitions between them. According to Definition 2.10, the discussed structure is still an LDBA (the determinism in the initial part is stronger than that required in the LDBA definition). An ε-transition allows an automaton to change its state without reading an input symbol. In practice, during an episode of the RL algorithm, the ε-transitions between $\mathcal{Q}_N$ and $\mathcal{Q}_D$ reflect the agent's "guess" on reaching $\mathcal{Q}_D$: accordingly, if after an ε-transition the associated labels in the accepting set of the automaton cannot be read, or the accepting states cannot be visited, then the guess is deemed to be wrong, and the trace is not accepted.*

2.3.1 Complexity Analysis of LTL Model-Checking

LTL model-checking in general, i.e. verifying if the given model satisfies the given property, is known to be equivalent to a satisfiability-checking problem. The model in our case is an MDP, and it has been shown that the satisfiability-checking problem for LTL is PSPACE-complete [174]. However, for the fragments in which only the eventual modality, i.e. "$\Diamond$" in Definition 2.6, is used the satisfiability problem is NP-complete. This is also the case for the next modality, i.e. "$\bigcirc$" in Definition 2.6, but interestingly when both $\Diamond$ and $\bigcirc$ are allowed the satisfiability problem is PSPACE-complete [175].

Despite PSPACE-complete hardness, the model-checkers or the satisfiability-checkers perform relatively well in practice [176]. This probably hints to the point that maybe even if this problem is PSPACE-complete "typical cases" of the problem in practice is tractable. Thus, a number of the modern model-checkers convert the MDP model-checking problem into a reachability problem and leverage DP. Such transformation result in a great improvement in time performance by essentially

consuming more space, which is used to store the results of the Bellman operation in each step. The proposed method in this dissertation, precisely targets this space consumption by focusing only on relevant parts of the state space, and therefore is able to offer exceptional time and space complexity.

2.4 Summary

We reviewed the basic concepts and notions in RL and LTL to establish a basis for the following chapters. The chapter discussed a selection of the most well-known RL algorithms in the field that have been successful in control policy synthesis. We formally introduced LTL as a formal language to specify the requirements and LDBA as a finite-state machine to express any LTL formula. As we see later, this allows us to shape a reward function for RL in a provably-correct manner, and also to provide guarantees on the optimality and convergence of our algorithm.

Chapter 3

Logically-Constrained Reinforcement Learning

3.1 Introduction

In this chapter, we discuss our proposed architecture Logically-Constrained Reinforcement Learning (LCRL), and the central ideas that will be the foundation for later chapters. In particular, we show that:

- LCRL automatically shapes a provably-correct reward that is specifically tailored to the given LTL property.
- LCRL optimal policy maximises the probability of LTL satisfaction.
- LCRL value function is able to determine the maximum probability of LTL satisfaction.
- LCRL is able to output the best possible control policy when the probability

of LTL satisfaction is zero.

Recall that the dynamics of the interaction of an RL agent with an unknown environment are often modelled as a black-box MDP. The LCRL architecture is the first model-free RL architecture that is applicable both over finite-state-action and uncountably infinite-state-action MDPs.

We are interested in synthesising a policy (or policies) for an unknown black-box MDP via RL such that the induced Markov chain satisfies a given LTL property with maximum probability. In order to explain the core concepts of the algorithm and for ease of exposition, let us assume for now that the MDP graph and the associated transition probabilities are known. Later these assumptions are entirely removed, and we stress that the algorithm can be run model-free. LCRL thus targets verified learning in its core, namely the model-free learning-based synthesis of policies that abide by a given LTL requirement. Computation of satisfaction probability of the traces associated to the synthesised policies is another key element of LCRL.

Further to this, in order to handle the general case of non-ergodic MDPs, LCRL consists of several resets, at each of which the agent is forced to re-start from the initial state of the MDP. Each reset defines an episode and as such the algorithm is called episodic RL.

3.2 MDP Augmentation

For now assume that the MDP structure and its transition kernel is known (Remark 3.1). We relate the MDP and the constructed LDBA by synchronising them, in order to create a new structure that is first of all compatible with RL and secondly that encompasses the given logical property.

Definition 3.1 (**Product MDP**). *Given an MDP* $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ *and an LDBA* $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ *with* $\Sigma = 2^{\mathcal{AP}}$*, the product MDP is defined as* $(\mathfrak{M} \otimes \mathfrak{A}) = \mathfrak{M}_{\mathfrak{A}} = (\mathcal{S}^{\otimes}, \mathcal{A}^{\otimes}, s_0^{\otimes}, P^{\otimes}, \mathcal{AP}^{\otimes}, L^{\otimes}, \mathcal{F}^{\otimes})$*, where* $\mathcal{S}^{\otimes} = \mathcal{S} \times \mathcal{Q}$*,* $s_0^{\otimes} = (s_0, q_0)$*,* $\mathcal{AP}^{\otimes} = \mathcal{Q}$*,* $L^{\otimes} : \mathcal{S}^{\otimes} \to 2^{\mathcal{Q}}$ *such that* $L^{\otimes}(s, q) = q$ *and* $\mathcal{F}^{\otimes} \subseteq \mathcal{S}^{\otimes}$ *is the set of accepting states* $\mathcal{F}^{\otimes} = \{F_1^{\otimes}, ..., F_f^{\otimes}\}$*, where* $F_j^{\otimes} = \mathcal{S} \times F_j$*. The transition kernel* $P^{\otimes}$ *is such that given the current state* (s_i, q_i) *and action* a*, the new state is* (s_j, q_j)*, where* $s_j \sim P(\cdot|s_i, a)$ *and* $q_j \in \Delta(q_i, L(s_j))$*. When the MDP* $\mathfrak{M}$ *has a finite state space, then* $P^{\otimes} : \mathcal{S}^{\otimes} \times \mathcal{A} \times \mathcal{S}^{\otimes} \to [0, 1]$ *is the transition probability function, such that* $(s_i \xrightarrow{a} s_j) \wedge (q_i \xrightarrow{L(s_j)} q_j) \Rightarrow P^{\otimes}((s_i, q_i), a, (s_j, q_j)) = P(s_i, a, s_j)$*. Furthermore, in order to handle* ε*-transitions we make the following modifications to the above definition of product MDP:*

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

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

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

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

Recall that an ε*-transition between* $\mathcal{Q}_N$ *and* $\mathcal{Q}_D$ *corresponds to a guess on reaching* $\mathcal{Q}_D$*. Hence, if after an* ε*-transition the associated labels in the accepting set of the automaton cannot be read, or the accepting states cannot be visited, then the guess was wrong, and the trace is not accepted.*

Note that LTL is a temporal language and satisfying an LTL property requires a policy that is non-Markovian and has an embedded memory [124, 125]. By

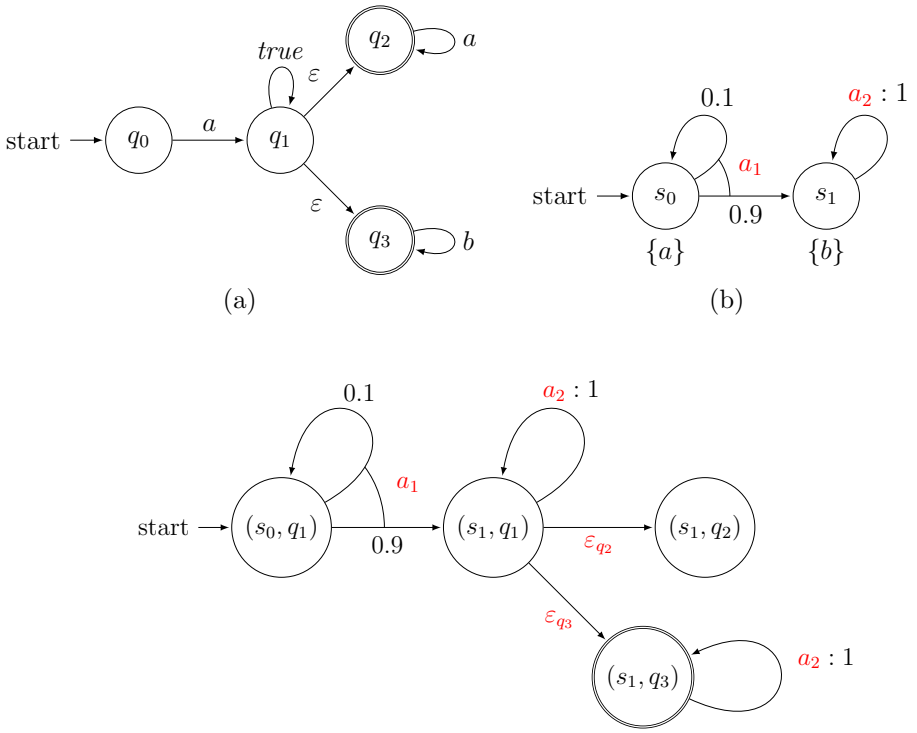


Figure 3.1: Example of product MDP: the top figures are the LDBA in Fig. 2.1 and an instance MDP, and the bottom one is the product of this MDP with the LDBA as per Definition 3.1. A crucial optimization proposed in [98] is that it is sufficient to take ε-transitions only from states in max end components of the product of $\mathfrak{M}$ and the initial partition of the LDGBA $\mathfrak{Q}_N$. Hence no ε-transitions have to be produced in the initial state of $\mathfrak{M}_\mathfrak{A}$.

constructing the product MDP we add an extra dimension to the state space of the original MDP, and that is the state space of the automaton representing the LTL formula. The role of the added dimension is to track LTL satisfaction and, hence, to synchronise the current state of the MDP with the state of the automaton: this essentially converts the non-Markovian LTL policy synthesis problem over the original MDP to a Markovian one over the product MDP. Furthermore, this allows to evaluate the (partial) satisfaction of the corresponding LTL property (or parts thereof). Fig. 3.1 illustrates the construction of a product MDP with the LDBA generated in Fig. 2.1.

Remark 3.1. *In order to clearly elucidate the role of different components in the proposed approach, we have employed model-dependent notions, such as transition probabilities and the product MDP. However, we emphasise that the proposed approach can run "model-free", and as such it does not depend on these components. In particular, as per Definition 2.10, the LDBA is composed of two disjoint sets of states $\mathcal{Q}_D$ (which is invariant) and $\mathcal{Q}_N$, where the accepting states belong to the set $\mathcal{Q}_D$. Since all transitions are deterministic within $\mathcal{Q}_N$ and $\mathcal{Q}_D$, the automaton transitions can be executed "only" by reading the labels, which makes the agent aware of the automaton state without explicitly constructing the product MDP. We will later define a reward function "on-the-fly", emphasising that the agent does not need to know the model structure or the transition probabilities (or their product).*

It is shown in [177] that for any MDP $\mathfrak{M}$ with non-Markovian reward, e.g. a time-dependent LTL-based reward, there exists a Markov reward MDP $\mathfrak{M}' = (\mathcal{S}', \mathcal{A}', s'_0, P', \mathcal{AP}', L')$ that is equivalent to $\mathfrak{M}$ such that the states of $\mathfrak{M}$ can be mapped to those of $\mathfrak{M}'$ where the corresponding states yield the same transition probabilities, and corresponding traces have the same rewards. Based on this result, [178] showed that the product MDP $\mathfrak{M}_\mathfrak{A}$ is indeed $\mathfrak{M}'$ defined above. Therefore, the non-Markovianity of the reward is resolved by synchronising the automaton with the original MDP, where the automaton represents the history of state labels that has led to that reward. This allows one to run RL over the product MDP and to find the optimal policy that maximises the corresponding Markovian reward. Before introducing a reward assignment for RL, we need to define the ensuing function. Recall that a generalised Büchi automaton accepts words that visit its accepting sets infinitely often.

Definition 3.2 (**Accepting Frontier Function**). *Let $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ be an LDBA where $\mathcal{F} = \{F_1, ..., F_f\}$ is the set of accepting conditions, and $F_j \subset \mathcal{Q}, 1 \leq j \leq f$. Define the function $Acc : \mathcal{Q} \times 2^\mathcal{Q} \to 2^\mathcal{Q}$ as the accepting frontier function,*

which executes the following operation over a given set $\mathbb{F} \subset 2^{\mathcal{Q}}$ *for every* $F_j \in \mathcal{F}$*:*

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

Once the state $q \in F_j$ *and the set* $\mathbb{F}$ *are introduced to the function Acc, it outputs a set containing the elements of* $\mathbb{F}$ *minus* F_j*. However, if* $\mathbb{F} = F_j$*, then the output is the family of sets of all accepting sets of the LDBA minus the set* F_j*. Finally, if the state* q *is not an accepting state then the output of Acc is* $\mathbb{F}$*. In short, the accepting frontier function excludes from* $\mathbb{F}$ *the accepting set that is currently visited, unless it is the only remaining accepting set. Otherwise, the output of* $Acc(q, \mathbb{F})$ *is* $\mathbb{F}$ *itself. What remains in* $\mathbb{F}$ *are those accepting sets that still need to be visited in order to attain the generalised Büchi accepting condition, as per Definition 2.9. Note that there is no need to produce all the subsets of accepting sets thanks to the* on-the-fly *execution of the accepting frontier function.*

3.3 Reward Shaping

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

following reward:

$$R(s^{\otimes}, a) = \begin{cases} r_p & \text{if } q' \in \mathbb{A},\ s^{\otimes\prime} = (s', q'), \\ r_n & \text{otherwise.} \end{cases} \tag{3.1}$$

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

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

Remark 3.2. *The set $\mathbb{A}$ always contains those accepting states that are needed to be visited at a given time: in this sense the reward function is "synchronous" with the accepting condition set by the LDBA. Thus, the agent is guided by the above reward assignment to visit these states and once all of the sets $F_k,\ k = 1, ..., f$, are visited, the accepting frontier $\mathbb{A}$ is reset. As such, the agent is guided to visit the accepting sets infinitely often, and consequently, to satisfy the given LTL property.*

The reward structure follows $r_p = M + y \times m \times rand(s^{\otimes})$ and $r_n = y \times m \times rand(s^{\otimes})$. The parameter $y \in \{0, 1\}$ is a constant, $0 < m \ll M$ are arbitrary positive values, and $rand : \mathcal{S}^{\otimes} \to (0, 1)$ is a function that generates a random number in $(0, 1)$ for each state $s^{\otimes}$ each time R is being evaluated. The role of the function $rand$ is to break the symmetry when neural nets are used for approximating the Q-function[1]. Also, note that parameter y acts as a switch to bypass the effect

[1]If all weights in a feedforward net start with equal values and if the solution requires that unequal weights be developed, the network can never learn. The reason is that the correlations between the weights within the same hidden layer can be described by symmetries in that layer, i.e. identical weights. Therefore, the neural net can generalise if such symmetries are broken and the redundancies of the weights are reduced. Starting with completely identical weights prevents the neural net from minimising these redundancies and from optimising the loss function [53].

of the *rand* function on R when no neural net is used. Thus, this switch is active $y = 1$ when the MDP state space is continuous, and disabled in others $y = 0$.

Remark 3.3. *Note that when running our algorithm there is no need to "explicitly build" the product MDP and to store all its states and transitions in memory. The automaton transitions can be executed on-the-fly as the agent reads the labels of the MDP states. Namely, the agent can track the automaton state by just looking at the trace that has been read so far. The agent only needs to store the current state of the automaton and observes the label at each step to check whether the automaton state has changed or not.*

Definition 3.3 (**G-sub-formula**)**.** *Given an LTL property φ and a set of G-sub-formulae $\mathcal{G}$[2] we define $\varphi[\mathcal{G}]$ the resulting formula when we substitute true for every G-sub-formula in $\mathcal{G}$ and $\neg$true for other G-sub-formulae of φ.*

Proposition 3.1. *Given an LTL formula φ and its associated LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta,)$, the accepting frontier set $\mathbb{A}$ is history-independent, namely the members of $\mathbb{A}$ only depend on the current state of the automaton and not on the sequence of automaton states that have been already visited.*

Proof. Let $\mathcal{G} = \{\Box\zeta_1, ..., \Box\zeta_f\}$ be the set of all G-sub-formulae of φ. Since elements of $\mathcal{G}$ are sub-formulae of φ we can assume an ordering over $\mathcal{G}$, so that if $\Box\zeta_i$ is a sub-formula of $\Box\zeta_j$, then $j > i$. In particular, $\Box\zeta_f$ is not a sub-formula any G-sub-formula.

The accepting component of the LDBA $\mathcal{Q}_D$ is a product of f of the DBAs $\{\mathbf{D}_1, ..., \mathbf{D}_f\}$ called G-monitors, such that each $\mathbf{D}_i = (\mathcal{Q}_i, q_{i0}, \Sigma, F_i, \delta_i)$ expresses $\Box\zeta_i[\mathcal{G}]$, where $\mathcal{Q}_i$ is the state space of the i-th G-monitor, $\Sigma = 2^{\mathcal{AP}}$, and $\delta_i : \mathcal{Q}_i \times \Sigma \to \mathcal{Q}_i$ [97]. Note that $\zeta_i[\mathcal{G}]$ has no G-sub-formula any more (Definition 3.3). The states of the G-monitor $\mathbf{D}_i$ are pairs of formulae where at each state the G-monitor

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

only checks if the run satisfies $\square\zeta_i[\mathcal{G}]$ while putting the next G-sub-formula in the ordering of $\mathcal{G}$ on hold, assuming that it is *true*.

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

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

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

As shown in [97], while a word w is being read by the accepting component of the LDBA, the set of G-sub-formulae that hold is "monotonically" expanding. If $w \in \mathit{Words}(\varphi)$, then eventually all G-sub-formulae become *true*. Now, let the current state of the automaton be $q_D = (q_1, ..., q_i, ..., q_f)$ while the automaton is checking whether $\square\zeta_i[\mathcal{G}]$ is satisfied or not, assuming that $\square\zeta_{i+1}$ is already *true* (though needs to be checked later), while all G-monitors $\square\zeta_j[\mathcal{G}],\ 1 \leq j \leq i-1$ have accepted w. At this point, the accepting frontier set $\mathbb{A} = \{F_i, F_{i+1}, ..., F_f\}$. Reasoning by contradiction, assume that the automaton returns to q_D but $\mathbb{A} \neq \{F_i, F_{i+1}, ..., F_f\}$, then at least one accepting set $F_j,\ j > i$ has been removed from $\mathbb{A}$. This means that $\square\zeta_j$ is a sub-formula of $\square\zeta_i$, violating the ordering of check on $\mathcal{G}$. This is a contradiction with respect to the ordering of G-sub-formulae. Thus, when the automaton returns to a particular state in $\mathcal{Q}_D$, the accepting frontier set $\mathbb{A}$ is always the same.

□

LCRL is a general control policy synthesis architecture that is adaptable to any off-the-shelf temporal difference RL algorithm with the discounted return objective. The `LCRL Core` in Fig. 3.2 is compatible with any proper RL scheme that conforms with the state and action space cardinality and dimension. Inside the LCRL box, the MDP state and the LDBA state are synchronised, resulting in an on-the-fly structure which we call a product MDP. In the following chapters we

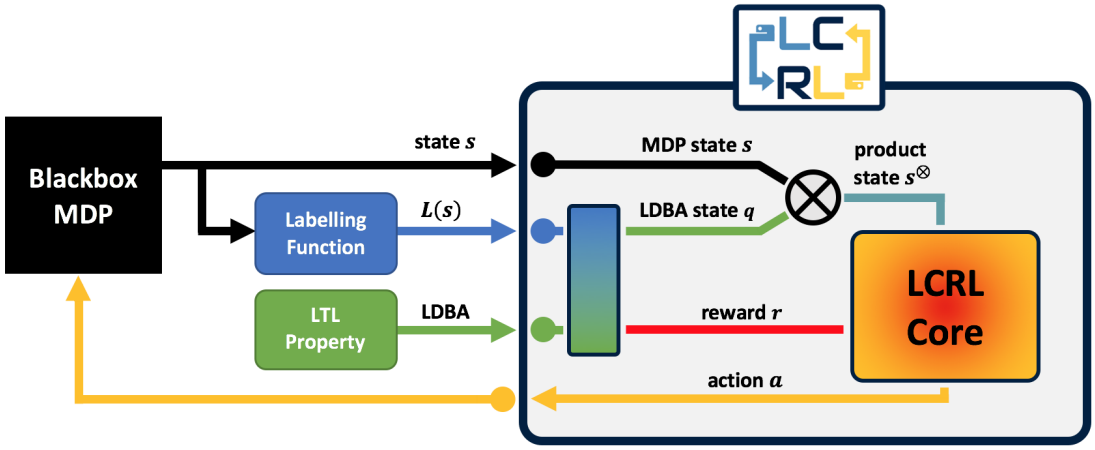


Figure 3.2: The LCRL general architecture.

discuss applicability and we present case studies to demonstrate the ease of use, and scalability of LCRL. Full instructions on how to import LCRL into Python are provided on a GitHub page that accompanies the distribution

`www.github.com/grockious/lcrl`

Theorem 3.1. *Let φ be the given LTL property and $\mathfrak{M}_\mathfrak{A}$ be the product MDP constructed by synchronising the MDP $\mathfrak{M}$ and the LDBA $\mathfrak{A}$ expressing φ. Then there exists a discounting factor close enough to* 1 *under which an optimal Markov policy on $\mathfrak{M}_\mathfrak{A}$ that maximises the expected return over the reward in* (3.1)*, also maximises the probability of satisfying φ. This optimal Markov policy induces a finite-memory policy on the MDP $\mathfrak{M}$.*

Proof. Assume that there exists a policy $\overline{\pi}$ that satisfies φ with maximum (non-zero) probability. Policy $\overline{\pi}$ induces a Markov chain $\mathfrak{M}_\mathfrak{A}^{\overline{\pi}}$ when it is applied over the MDP $\mathfrak{M}_\mathfrak{A}$. This Markov chain comprises a disjoint union between a set of transient states $\mathfrak{T}_{\overline{\pi}}$ and h sets of irreducible recurrent classes $\mathfrak{R}_{\overline{\pi}}^i$, $i = 1, ..., h$ [162], namely:

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

From (2.11), policy $\overline{\pi}$ satisfies φ if and only if:

$$\exists \mathfrak{R}^i_{\overline{\pi}} \text{ s.t. } \forall j \in \{1, ..., f\},\ F^{\otimes}_j \cap \mathfrak{R}^i_{\overline{\pi}} \neq \emptyset. \tag{3.2}$$

The recurrent classes that satisfy (3.2) are called accepting. From the irreducibility of the recurrent class $\mathfrak{R}^i_{\overline{\pi}}$ we know that all the states in $\mathfrak{R}^i_{\overline{\pi}}$ communicate with each other thus, once a trace ends up in such set, all the accepting sets are going to be visited infinitely often. Therefore, from the definition of $\mathbb{A}$ and of the accepting frontier function (Definition 3.2), the agent receives a positive reward r_p ever after it has reached an accepting recurrent class $\mathfrak{R}^i_{\overline{\pi}}$.

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

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

$$\forall j \in \{1, ..., f\},\ F^{\otimes}_j \cap \mathfrak{R}^k_{\overline{\pi}} = \emptyset;$$

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

$$\exists J \subset 2^{\{1,...,f\}} \setminus \{1, ..., f\} \text{ s.t. } \forall j \in J,\ F^{\otimes}_j \cap \mathfrak{R}^k_{\overline{\pi}} \neq \emptyset. \tag{3.3}$$

In the second case, the agent is able to visit some accepting sets but not all of them. This means that in the update rule of the frontier accepting set $\mathbb{A}$ in Definition 3.2, the case where $(q \in F_j) \wedge (\mathbb{A} = F_j)$ will never happen since there exist always at least one accepting set that has no intersection with $\mathfrak{R}^k_{\overline{\pi}}$. Therefore, after a limited number of times, no positive reward can be obtained, and the reinitialisation of $\mathbb{A}$ in Definition 3.2 is blocked.

Recall Definition 2.3, where the expected return for a state $s^{\otimes} \in \mathcal{S}^{\otimes}$ is defined

as in (2.2) and (2.3):

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

In both cases, from (3.1), for any arbitrary $r_p > 0$ (and $r_n = 0$), there always exists a discounting coefficient η such that the expected return of a trace hitting $\mathfrak{R}^i_{\overline{\pi}}$ with unlimited number of successive times attaining positive reward, is higher than the expected return of any other trace. With unlimited number of obtaining positive reward for the traces entering the accepting recurrent class $\mathfrak{R}^i_{\overline{\pi}}$, and with a state-dependent discount factor, it can be shown that the expected reward is bounded and is higher than that for non-accepting traces, which have limited number of attainment of positive rewards.

In the following, by contradiction, we show that any optimal policy π^* which optimises the expected return will satisfy the property with maximum probability if η is close to one. Recall from Definition 2.7 that the probability of satisfying φ under policy π at state $s^{\otimes}$ is:

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

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

$$\overline{\pi} = \underset{\pi \in \mathcal{D}}{\operatorname{argsup}}\, Pr(s^{\otimes}..^{\pi} \models \varphi). \tag{3.4}$$

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

$$U^{\overline{\pi}}(s^{\otimes}) = \mathbb{E}^{\overline{\pi}}\Big[\sum_{n=0}^{\infty}\gamma(s_n^{\otimes})^{N(s_n^{\otimes})}\ r_n\Big|s_0^{\otimes}=s^{\otimes},\ \rho=L(s_0)L(s_1)...\models\varphi\Big]Pr(s^{\otimes}..^{\overline{\pi}}\models\varphi)$$
$$+\ \mathbb{E}^{\overline{\pi}}\Big[\sum_{n=0}^{\infty}\gamma(s_n^{\otimes})^{N(s_n^{\otimes})}\ r_n\Big|s_0^{\otimes}=s^{\otimes},\ \rho=L(s_0)L(s_1)...\not\models\varphi\Big]Pr(s^{\otimes}..^{\overline{\pi}}\not\models\varphi). \tag{3.5}$$

Of course, when the policy traces satisfy the property, with unlimited number of positive reward attainment the expected return under (2.2) is

$$\mathbb{E}^{\overline{\pi}}\Big[\sum_{n=0}^{\infty}\gamma(s_n^{\otimes})^{N(s_n^{\otimes})}\ r_n\Big|s_0^{\otimes}=s^{\otimes},\ \rho=L(s_0)L(s_1)...\models\varphi\Big]Pr(s^{\otimes}..^{\overline{\pi}}\models\varphi)= \tag{3.6}$$
$$\frac{r_p}{1-\eta}Pr(s^{\otimes}..^{\overline{\pi}}\models\varphi).$$

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

$$\mathbb{E}^{\overline{\pi}}\Big[\sum_{n=0}^{\infty}\gamma(s_n^{\otimes})^{N(s_n^{\otimes})}\ r_n\Big|s_0^{\otimes}=s^{\otimes},\ \rho=L(s_0)L(s_1)...\not\models\varphi\Big]Pr(s^{\otimes}..^{\overline{\pi}}\not\models\varphi)= \tag{3.7}$$
$$\sum_{\rho\in L(s^{\otimes}..^{\overline{\pi}}\not\models\varphi)}Pr(\rho\in L(s^{\otimes}..^{\overline{\pi}}\not\models\varphi))\frac{r_p(1-\eta^{|\overline{J}|_\rho})}{1-\eta},$$

where $|\overline{J}|_\rho$ is the (finite) number of times that the trace ρ intersected with the accepting frontier set $\mathbb{A}$ and the agent received a positive reward (see (3.3)). From (3.6) and (3.7) we can reformulate (3.5) as follows:

$$U^{\overline{\pi}}(s^{\otimes}) = \frac{r_p}{1-\eta}Pr(s^{\otimes}..^{\overline{\pi}}\models\varphi)+\sum_{\rho\in L(s^{\otimes}..^{\overline{\pi}}\not\models\varphi)}Pr(\rho\in L(s^{\otimes}..^{\overline{\pi}}\not\models\varphi))\frac{r_p(1-\eta^{|\overline{J}|_\rho})}{1-\eta}. \tag{3.8}$$

Similarly for the optimal policy π^* we have

$$U^{\pi^*}(s^{\otimes}) = \frac{r_p}{1-\eta} Pr(s^{\otimes}..^{\pi^*} \models \varphi) + \sum_{\rho \in L(s^{\otimes}..^{\pi^*} \not\models \varphi)} Pr(\rho \in L(s^{\otimes}..^{\pi^*} \not\models \varphi)) \frac{r_p(1-\eta^{|J^*|_\rho})}{1-\eta}, \tag{3.9}$$

where $|J^*|_\rho$ is the (finite) number of times that the trace ρ intersected with the accepting frontier set $\mathbb{A}$. We then factorise $r_p/1-\eta$ from (3.8) and (3.9):

$$U^{\overline{\pi}}(s^{\otimes}) = \frac{r_p}{1-\eta}\Big[Pr(s^{\otimes}..^{\overline{\pi}} \models \varphi) + \sum_{\rho \in L(s^{\otimes}..^{\overline{\pi}} \not\models \varphi)} Pr(\rho \in L(s^{\otimes}..^{\overline{\pi}} \not\models \varphi))(1-\eta^{|\overline{J}|_\rho})\Big]. \tag{3.10}$$

$$U^{\pi^*}(s^{\otimes}) = \frac{r_p}{1-\eta}\Big[Pr(s^{\otimes}..^{\pi^*} \models \varphi) + \sum_{\rho \in L(s^{\otimes}..^{\pi^*} \not\models \varphi)} Pr(\rho \in L(s^{\otimes}..^{\pi^*} \not\models \varphi))(1-\eta^{|J^*|_\rho})\Big]. \tag{3.11}$$

Now suppose that the optimal policy π^* does not satisfy the property φ with maximum probability. Given that $\overline{\pi}$ maximises the satisfaction probability, from (3.4) we have:

$$Pr(s^{\otimes}..^{\overline{\pi}} \models \varphi) > Pr(s^{\otimes}..^{\pi^*} \models \varphi). \tag{3.12}$$

At the same time, it is easy to see from (3.10) and (3.11) that:

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

and consequently from (3.12):

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

This is, however, in direct contrast with Definition 2.4 and the optimality of the policy π^*, leading to a contradiction. This essentially means that the optimal policy π^* maximises the probability of satisfying φ. □

Remark 3.4. *Note that the projection of policy π^* onto the state space of the original MDP $\mathfrak{M}$ yields a finite memory policy $\pi^*_{\mathfrak{M}}$. Thus, if the generated traces under π^* maximise the LTL satisfaction probability, so do the traces of $\pi^*_{\mathfrak{M}}$.*

Remark 3.5. *Note that the optimality of the policies generated using (deep) neural-network-based approaches, depends on a number of factors, such as the network structure, number of hidden layers, and activation functions. Specifically, convergence guarantees of such methods to a true optimal policy is not well developed and quantification of the sub-optimality of the policy generated by these methods is out of the scope of this work.*

An interesting extension of Theorem 3.1 is the following corollary.

Corollary 3.1 (**Maximum Probability of Satisfaction**). *Under the assumptions of Theorem 3.1 and from Definition 2.7, for a discounting factor close enough to* 1 *the maximum probability of satisfaction at any state $s^{\otimes}$ can be determined from LCRL value function:*

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

Proof. The proof is a direct consequent of (3.11) and Theorem 3.1 results when $\eta \to 1^-$. □

Definition 3.4 (**Closeness to Satisfaction**). *Assume that the probability of satisfying the property φ for two policies π_1 and π_2 is zero. Accordingly, there are accepting sets in the automaton that have no intersection with runs of induced Markov chains $\mathfrak{M}^{\pi_1}$ and $\mathfrak{M}^{\pi_2}$. We say that π_1 is closer to satisfying the property*

if runs of $\mathfrak{M}^{\pi_1}$ cross a larger number of distinct accepting sets of the automaton, than runs of $\mathfrak{M}^{\pi_2}$.

Corollary 3.2. *If no policy in the finite-state finite-action MDP $\mathfrak{M}$ can be generated to satisfy the property φ, LCRL yields in the limit a policy that is closest (according to the previous Definition) to satisfying the given LTL formula φ.*

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

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

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

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

which means that there are some automaton accepting sets like F_j that cannot be visited. Therefore, after a limited number of times no positive reward is given by the reward function $R(s^{\otimes}, a)$. However, the closest recurrent class to satisfying the property is the one that intersects with more distinct accepting sets.

By Definition 2.3, for any arbitrary $r_p > 0$ (and $r_n = 0$), the expected return at the initial state for a trace with highest number of intersections with distinct accepting sets is maximum among other traces. Hence, by the convergence guarantees of QL, the optimal policy produced by LCRL converges to a policy whose recurrent classes of its induced Markov chain have the highest number of intersections with the accepting sets of the automaton. □

3.4 Summary

We consider an RL problem in which we exploit the structural information provided by the LTL specification, and by constructing an optimal Markov policy for each state of the associated LDBA. Our proposed approach learns a satisfying policy without requiring any information about the grounding of the LTL task to be explicitly specified. Namely, the labelling assignment in Definition 2.1 is unknown a-priori, and the algorithm solely relies on experience samples gathered on-the-fly. Given an LTL mission task and an unknown black-box MDP, we proposed the first model-free RL method to constrain resulting traces by using an LTL property. We have argued that converting the LTL property to a GBA-based LDBA results in a significantly smaller automaton than to a DRA (the standard approach in the literature), which decreases the size of the product MDP and increases RL convergence rate. A DRA needs more complicated accepting conditions than a Büchi automaton; thus, a more complex reward assignment is required. 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.

Chapter 4

Model-free Tabular RL under LTL

4.1 Introduction

In this chapter, we discuss and show that LCRL, a model-free RL architecture with discounted reward, can be employed for (1) LTL policy synthesis, and (2) quantitative model-checking in finite MDPs. This chapter also includes proofs of optimality and convergence of LCRL when the MDP is finite.

Recall that the simplest case in Definition 2.1 is when the MDP state space and action space are both countably finite. This case, however, covers a significant number of control applications. Over finite-state finite-action MDPs, as introduced in the last chapter, we can directly run QL over the product MDP $\mathfrak{M}_{\mathfrak{A}}$ with the reward shaping proposed in (3.1), where we have set $y = 0$. In order to also handle non-ergodic MDPs, we propose to employ a variant of standard QL that consists of

several resets, at each of which the agent is forced to re-start from its initial state s_0. Each reset defines an episode, as such the algorithm is called "episodic QL". However, for the sake of simplicity, we omit the term "episodic" in the rest of the dissertation and we use the term Logically-Constrained QL (LCQL).

4.2 Logically-Constrained QL

QL [33], is the most widely-used RL algorithm for solving MDPs from simple control tasks to the basis of more recent developments in RL. As stated earlier, QL is proved to converge to the optimal Q-function [31, 33], and thus we are able to synthesise the optimal policy in the limit. The optimal policy produced by LCQL indeed satisfies the given LTL property (Definition 2.8) with maximum probability possible (Definition 2.7) as per Theorem 3.1.

Remark 4.1. *Note that LCQL outputs its policy by choosing the maximum Q-value at any given state. Further, as shown in Corollary 3.1, we can derive the probability of satisfying the LTL property from the Q-values. This means that, if there exists more than one optimal policy, i.e. if there is more than one satisfying policy corresponding to the same probability, then LCQL is able to find all of them by presenting the same Q-values to the agent for these policies. Thus, the agent is free to choose between these policies by arbitrarily choosing actions that have the same expected return.*

We have discussed a model-free RL implementation that is capable of synthesising policies (when existing) that maximally satisfy an LTL formula over a finite-state finite-action MDP. In the following, we present an additional component, which allows us to quantify the quality of the resulting policy by calculating the probability of satisfaction associated with the policy π^*.

4.2.1 Probability of Satisfaction of a Property

The Probability of Satisfaction of a Property (PSP) can be calculated via standard DP, as implemented for instance in PRISM [179] and Storm [180]. However, as discussed before, DP is quite limited when the state space of the given MDP is large, and as assumed in this work, when the transition probability matrix is unknown.

In the following, on top of the LCRL internal mechanism for calculating the maximum probability of satisfaction, we propose a local value iteration method as part of LCQL that calculates this probability in parallel with RL. RL guides the local update of the value iteration, such that it only focuses on parts of the state space that are relevant to the satisfaction of the property. This allows the value iteration to avoid an exhaustive sweep over the whole state space of the MDP and thus to converge faster.

Recall that the transition probability function $P^{\otimes}$ is not known. Further, according to Definition 3.1, for any two MDP states $s_i, s_j \in \mathcal{S}$ and any pair of automaton states $q_i, q_j \in \mathcal{Q}$, $P^{\otimes}((s_i, q_i), a, (s_j, q_j)) = P(s_i, a, s_j)$ if $(s_i \xrightarrow{a} s_j)$ and $(q_i \xrightarrow{L(s_j)} q_j)$, showing the intrinsic dependence of $P^{\otimes}$ on P. This allows to apply the definition of α-approximation in MDPs [46] as follows.

Let us introduce two functions $\Psi : \mathcal{S} \times \mathcal{A} \to \mathbb{N}$ and $\psi : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{N}$ over the MDP $\mathfrak{M}$. Function $\psi(s, a, s')$ represents the number of times the agent executes action a in state s, thereafter moving to state s', whereas $\Psi(s, a) = \sum_{s' \in \mathcal{S}} \psi(s, a, s')$. The maximum likelihood of $P(s, a, s')$ is a Dirichlet distribution with the mean $\bar{P}(s, a, s') = \psi(s, a, s')/\Psi(s, a)$, so that the variance of this distribution asymptotically converges to zero and $\bar{P}(s, a, s') = P(s, a, s')$. Function $\Psi(s, a)$ is initialised to be equal to 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), and function $\psi(s, a, s')$ is initialised to be equal to zero.

Definition 4.1 (**Non-accepting Sink Component**). *A non-accepting sink component of the LDBA* $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ *is a directed graph induced by a set of states* $Q \subset \mathcal{Q}$ *such that (1) the graph is strongly connected; (2) it does not include all accepting sets* $F_k,\ k = 1, ..., f$*; and (3) there exist no other strongly connected set* $Q' \subset \mathcal{Q},\ Q' \neq Q$ *that* $Q \subset Q'$*. We denote the union of all non-accepting sink components of* $\mathfrak{A}$ *as* $\mathbb{N}$*.*

The set $\mathbb{N}$ are those components in the automaton that are surely not accepting and impossible to escape from. Thus, reaching them is equivalent to not being able to satisfy the given LTL property. Now, consider a function $PSP : \mathcal{S}^{\otimes} \to [0, 1]$. For a given state $s^{\otimes} = (s, q)$, the PSP function is initialised as $PSP(s^{\otimes}) = 0$ if q belongs to $\mathbb{N}$. Else, it is initialised as $PSP(s^{\otimes}) = 1$.

Definition 4.2 (**Optimal PSP**). *The optimal PSP vector is denoted by* $\underline{PSP}^* = (PSP^*(s_1), ..., PSP^*(s_{|\mathcal{S}|}))$*, where* $PSP^* : \mathcal{S}^{\otimes} \to [0, 1]$ *is the optimal PSP function and* $PSP^*(s_i)$ *is the optimal PSP value starting from state* s_i *such that*

$$PSP^*(s_i) = \sup_{\pi \in \mathcal{D}} PSP^{\pi}(s_i),$$

where $PSP^{\pi}(s_i)$ *is the PSP value of state* s_i *if we use the policy* π *to determine subsequent states.*

In the following, we prove that a proposed update rule that makes PSP converge to PSP^*.

Definition 4.3 (**Bellman operation [31]**). *For any vector such as* $\underline{PSP} = (PSP(s_1), ..., PSP(s_{|\mathcal{S}|}))$ *in the MDP* $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$*, the Bellman DP operation* T *over the elements of* $\underline{PSP}$ *is defined as:*

$$T\ PSP(s_i) = \max_{a \in \mathcal{A}_{s_i}} \sum_{s' \in \mathcal{S}} P(s_i, a, s') PSP(s'). \tag{4.1}$$

If the operation T is applied over all the elements of $\underline{PSP}$, we denote it as $T\ \underline{PSP}$.

Proposition 4.1 (From [31]). *The optimal PSP vector $\underline{PSP}^*$ satisfies the following equation:*

$$\underline{PSP}^* = T\ \underline{PSP}^*,$$

and additionally, $\underline{PSP}^*$ is the only solution of the equation $\underline{PSP} = T\ \underline{PSP}$, i.e., the solution is unique.

In the standard value iteration method, the value estimation is simultaneously updated for all states. The proposed alternative method is to update the value for one state at a time. This method is known as asynchronous value iteration.

Definition 4.4 (**Gauss-Seidel Asynchronous Value Iteration (AVI) [31]**). *We denote AVI operation by F and is defined as follows:*

$$F\ PSP(s_1) = \max_{a \in \mathcal{A}} \{ \sum_{s' \in \mathcal{S}} P(s_1, a, s') PSP(s') \}, \tag{4.2}$$

where s_1 is the state that current state at the MDP, and for all $s_i \neq s_1$:

$$F\ PSP(s_i) = \max_{a \in \mathcal{A}} \left[\sum_{s' \in \mathcal{S}_L} P(s_i, a, s') F\ PSP(s') + \sum_{s' \in \mathcal{S}_R} P(s_i, a, s') PSP(s') \right], \tag{4.3}$$

where $\mathcal{S}_L = \{s_1, ..., s_{i-1}\}$ and $\mathcal{S}_R = \{s_i, ..., s_{|\mathcal{S}|}\}$. By (4.3) we update the value of PSP state by state and use the calculated value for the next step.

Proposition 4.2 (From [31]). *Let $k_0, k_1, ...$ 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 $m = 0, 1, ...$ Then the sequence of value vectors generated by AVI asymptotically converges to $\underline{PSP}^*$.*

Lemma 4.1 (From [31]). *The operator F is a contraction mapping with respect to the infinity norm. In other words, for any two value vectors PSP and PSP′:*

Algorithm 4.1: Logically-Constrained QL

input : LTL specification, *it_threshold*, γ, μ
output : π^* and $\underline{PSP}^*$

initialize $Q : \mathcal{S}^{\otimes} \times \mathcal{A}^{\otimes} \to \mathbb{R}_0^+$
initialize $PSP : \mathcal{S}^{\otimes} \to [0, 1]$
initialize $\psi : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{N}$
initialize $\Psi : \mathcal{S} \times \mathcal{A} \to \mathbb{N}$
convert the desired LTL property to LDBA $\mathfrak{A}$
initialize $\mathbb{A}$
initialize *episode-number* $:= 0$
initialize *iteration-number* $:= 0$
while Q *is not converged* **do**
  episode-number $++$
  $s^{\otimes} = (s_0, q_0)$
  while $(q \notin \mathbb{N} : \ s^{\otimes} = (s, q))$ & (*iteration-number* $<$ *it_threshold*) **do**
    iteration-number $++$
    choose $a_* = \pi(s^{\otimes}) = \arg\max_{a \in \mathcal{A}} Q(s^{\otimes}, a)$ # or $\epsilon-$greedy
    $\Psi(s^{\otimes}, a_*) ++$
    move to $s^{\otimes}_* = s^{\otimes}_* = (s_*, q_*)$ by a_*
    if $\Psi(s^{\otimes}, a_*) = 2$ **then**
      $\psi(s^{\otimes}, a_*, s^{\otimes}_*) = 2$
    else
      $\psi(s^{\otimes}, a_*, s^{\otimes}_*) ++$
    end
    receive the reward $R(s^{\otimes}, a_*)$
    $\mathbb{A} \leftarrow Acc(q_*, \mathbb{A})$
    $Q(s^{\otimes}, a_*) \leftarrow Q(s^{\otimes}, a_*) + \mu[R(s^{\otimes}, a_*) - Q(s^{\otimes}, a_*) + \gamma(s^{\otimes}) \max_{a'}(Q(s^{\otimes}_*, a'))]$
    $\bar{P}^{\otimes}(s^{\otimes}_i, a, s^{\otimes}_j) \leftarrow \psi(s^{\otimes}_i, a, s^{\otimes}_j) / \Psi(s^{\otimes}_i, a), \ \forall s^{\otimes}_i, s^{\otimes}_j \in \mathcal{S}^{\otimes}$ and $\forall a \in \mathcal{A}$
    $PSP(s^{\otimes}) \leftarrow \max_{a \in \mathcal{A}} \sum_{s^{\otimes\prime} \in \mathcal{S}^{\otimes}} \bar{P}^{\otimes}(s^{\otimes}, a, s^{\otimes\prime}) \times PSP(s^{\otimes\prime})$
    $s^{\otimes} = s^{\otimes}_*$
  end
end

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

Proposition 4.3 (Convergence, [46])**.** *From Lemma 4.1, and under the assumptions of Proposition 4.2, $\bar{P}(s, a, s')$ converges to $P(s, a, s')$, and from Proposition 4.2, the AVI value vector $\underline{PSP}$ asymptotically converges to $\underline{PSP}^*$, i.e. the probability that could be alternatively calculated by DP-based methods if the MDP was fully known.*

We conclude this section by presenting the overall procedure in Algorithm 4.1. The input of LCQL includes *it_threshold*, which is an upper bound on the number of iterations in each learning episode. Within each episode, the agent either ends up in $\mathbb{N}$, after which the LTL property cannot be satisfied and a new learning episode is required, or uses the *iteration-number* up to its upper bound *it_threshold*.

Recall that the function $\psi(s, a, s')$ represents the number of times the agent executes action a in state s, thereafter moving to state s', and $\Psi(s, a) = \sum_{s' \in \mathbb{S}} \psi(s, a, s')$. Therefore, at any given time the best estimate of the agent for $P_a(s, a, s')$ is the mean $\psi(s, a, s')/\Psi(s, a)$. Function $\Psi(s, a)$ is initialised to be 1 for every state-action pair, reflecting the fact that at any given state it is possible to take any action. The function $\psi(s, a, s')$ is initialised to 0. Once, the transition (s, a, s') is taken for the first time, $\Psi(s, a) \leftarrow 2$, and thus $\psi(s, a, s')$ has to be incremented to 2 to reflect the new belief $P_a(s, a, s') = 1$ (Algorithm 4.1, lines 17-21).

In the following, we describe another contribution of this work in dealing with time-varying MDPs that show periodic behaviours. Such behaviours can be seen in a number of applications such as physical systems and video games.

4.2.2 Transfer Learning for Time-Varying MDPs

The classical RL setting dealing with static (time-invariant) MDPs is not particularly representative of real-world situations where the environment is time-varying. To this end, we consider MDPs that exhibit time-dependent behaviours. This was inspired by the initial chamber of Atari 2600 Montezuma's Revenge, where a skull rolls periodically on a platform. Such time-varying obstacles can easily render QL useless, as the method is memory-less and only takes into consideration the current state when choosing the best action.

Periodic behaviour can be encompassed in the MDP dynamics by extending its state space with a time variable. However, this approach is in general not

computationally viable due to state-space explosion, which is caused by adding the extra time dimension. More specifically, for each time-step in the period of the MDP dynamics, the agent finds itself in a completely new environment and has to learn everything anew.

To overcome this limitation we model the periodically moving obstacle as a Kripke structure. The structure over $\mathcal{AP}^{\otimes}$, which comprises the possible positions of the obstacle, when unfolded to account for directionality (as shown in Fig. 4.1), represents the obstacle positions. It can be defined as $\mathfrak{D}(\mathcal{K}, k_0, \Delta', L')$ where $\mathcal{K}$ is the state space with transition relation $\Delta' \subseteq \mathcal{K} \otimes \mathcal{K}$ and labelling function $L' : \mathcal{K} \to 2^{\mathbb{N}}$. In this dissertation, for the sake of simplicity, we assume that the labelling function assigns a natural number to each state.

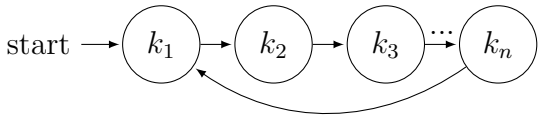


Figure 4.1: Kripke structure capturing the time-varying behaviour of the MDP.

We propose to first take the cross product with the generated LDBA $\mathfrak{A}$ (Definition 3.1) and Kripke structure $\mathfrak{D}$ (notice again that we do not explicitly product the Kripke structure with the MDP). The resulting structure is a time-varying automaton with which we can synchronise the original MDP $\mathfrak{M}$ on-the-fly (see Remark 3.3). Thanks to this new automaton and the fact that no explicit product MDP is constructed and stored in memory, the proposed approach can handle the time-dependent behaviour of the MDP much more efficiently as opposed to an explicit encoding of time as lifting to the MDP. Furthermore, thanks to the local asynchronous update of model-free RL we do not need to execute the Bellman operator over the whole state space as in DP.

A related idea has been recently discussed in [181], where a Kripke-like structure has been proposed to synthesise a controller for partially-observable MDPs: however,

whereas the finite-state machine in [181] is designed to act as a memory for the sequence of observations, the Kripke structure in this work is employed to capture the periodic dynamics of the MDP. The final product captures the time-varying parts of the original MDP, and maps it to a static MDP including $\mathfrak{D}$. The following definition formalises the intuition:

Definition 4.5 (**Periodic Product MDP**). *Given an MDP* $\mathfrak{M}(\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$, *an LDBA* $\mathfrak{A}(\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ *with* $\Sigma = 2^{\mathcal{AP}}$, *and a Kripke structure* $\mathfrak{D}(\mathcal{K}, k_0, \Delta', L')$, *the product MDP is defined as* $\mathfrak{M}_{\mathfrak{AD}}(\mathcal{S}^{\boxtimes}, \mathcal{A}, s_0^{\boxtimes}, P^{\boxtimes}, \mathcal{AP}^{\boxtimes}, L^{\boxtimes}, \mathcal{F}^{\boxtimes})$, *where* $\mathcal{S}^{\boxtimes} = \mathcal{S} \times (\mathcal{Q} \times \mathcal{K})$, $s_0^{\boxtimes} = (s_0, q_0, k_0)$, $\mathcal{AP}^{\boxtimes} = \mathcal{Q}$, $L^{\boxtimes} : \mathcal{S}^{\boxtimes} \to 2^{\mathcal{Q}}$ *such that* $L^{\boxtimes}(s, q, k) = q$ *and* $\mathcal{F}^{\boxtimes} \subseteq \mathcal{S}^{\boxtimes}$ *is the set of accepting states* $\mathcal{F}^{\boxtimes} = \{F_1^{\boxtimes}, ..., F_f^{\boxtimes}\}$ *where* $F_j^{\otimes} = \mathcal{S} \times F_j \times \mathcal{K}$. *The intuition behind the transition kernel* $P^{\boxtimes}$ *is that given the current state* (s_i, q_i, k_i) *and action* a, *the new state is* (s_j, q_j, k_j) *where* $s_j \sim P(\cdot|s_i, a)$, $q_j \in \Delta(q_i, L(s_j))$, *and* $k_j \in \Delta'(k_i)$. *When the MDP* $\mathfrak{M}$ *is finite-state then* $P^{\boxtimes} : \mathcal{S}^{\boxtimes} \times \mathcal{A} \times \mathcal{S}^{\boxtimes} \to [0, 1]$ *is the transition probability function such that* $(s_i \xrightarrow{a} s_j) \wedge (q_i \xrightarrow{L(s_j)} q_j) \wedge (k_i \to k_j) \Rightarrow P^{\boxtimes}((s_i, q_i, k_i), a, (s_j, q_j, k_j)) = P(s_i, a, s_j)$.

The curse of dimensionality related to the alternative explicit encoding of the periodicity within the MDP is now turned into the slowness of the learning process. To speed up the process we can use the observation that the agent, and inherently the Q-values, are only affected by the time-varying parts of the original MDP when they are in close proximity. In other words, the optimal strategy can be learnt more quickly by sharing the state-action values across the dimension defined by $\mathcal{K}$.

Recall the classical update rule in QL, when the agent executes action a at state $s^{\boxtimes} = (s, q, k)$ is as follows:

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

Once a positive behaviour is learned in dealing with the time-varying part, it is

propagated along the $\mathcal{K}$ dimension to allow the agent to do the same behaviour in states that are in close proximity of the time-varying part. Technically speaking, when the agent executes action a at state $s^{\boxtimes} = (s, q, k)$ and updates the Q-value for $s^{\boxtimes}, a$, then

$$\forall k' \in \mathcal{K} \setminus \{k\},\ Q(s, q, k', a) \leftarrow \max\{Q(s, q, k, a), Q(s, q, k', a)\}.$$

This update rule, once combined with the QL classic update rule, allows the positive behaviours to be echoed in $\mathcal{S}^{\boxtimes}$ and significantly reduces the learning time.

In other words, the knowledge that the agent has gained while solving one time instance is transferred and applied to different but related times instances. The insight behind the use of transfer learning in this work is to generalise over components of the LTL task, i.e. the automaton states, and across time. Namely, the Q-value update for a given state-action pair affects the Q-values of distinct but related state-action pairs. Transfer learning is used to improve the learning process for one specific time instance in the automaton by transferring information from other instances. The general idea of transfer learning has been applied over different domains in supervised [182–184] and unsupervised learning [181, 185, 186].

4.3 Experimental Results

We discuss a number of planning experiments dealing with policy synthesis problems around temporal specifications that are extended with safety requirements, when the state space is finite.

The first experiment is an LTL-constrained control synthesis problem for a robot in a slippery grid-world. Let the grid be an $L \times L$ 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 $\mathcal{A} = \{\mathit{left}, \mathit{right}, \mathit{up}, \mathit{down}, \mathit{stay}\}$ by

which the robot is able to move to other states (e.g. s') with the probability of $P(s, a, s'), a \in \mathcal{A}$. 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, 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). This example is a well-known benchmark and is often referred to as "slippery grid-world". This experiment has 1600 states, with 5 enabled actions at each 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. We consider two 40×40 regions and one 3×3 region with different labels as in Fig. 4.2. In Region 3 and in the 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 and the agent has to avoid the top row otherwise it gets trapped.

In the first experiment, we consider the following LTL properties. The first two properties (4.4) and (4.5) focus on safety and reachability, while the third property (4.6) requires a sequential visit to states with label p and then to target t:

$$\Diamond t \wedge \Box(t \to \Box t) \wedge \Box(u \to \Box u), \tag{4.4}$$

$$\Diamond \Box t, \tag{4.5}$$

and

$$\Diamond(p \wedge \Diamond t) \wedge \Box(t \to \Box t) \wedge \Box(u \to \Box u), \tag{4.6}$$

where t stands for "target", u stands for "unsafe", and p refers to the area that has to be visited before visiting the area with label t. Property (4.4) asks the agent

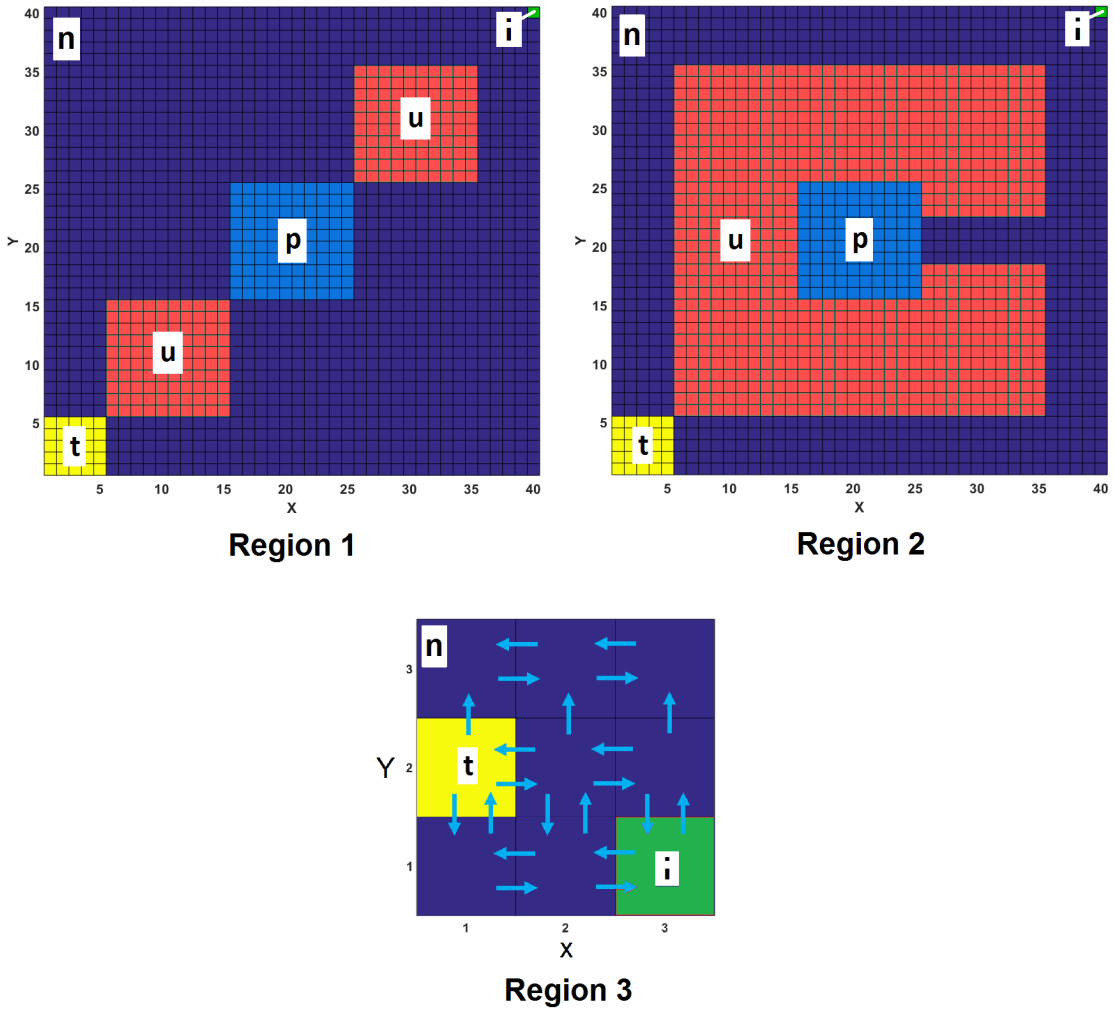


Figure 4.2: Slippery grid-world with $|\mathcal{S}| = 1600$ - green: initial state (i); dark blue: neutral (n); red: unsafe (u); light blue: pre-target (p); yellow: target (t).

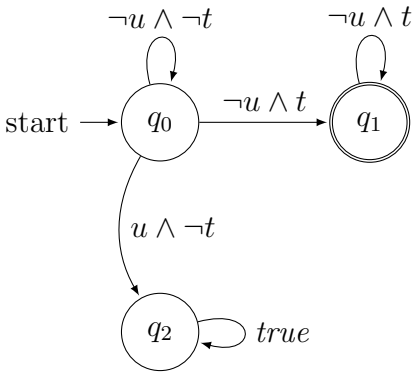


Figure 4.3: LDBA for the specification in (4.4) with removed transitions labelled $t \wedge u$ (since it is impossible to be at target and unsafe at the same time).

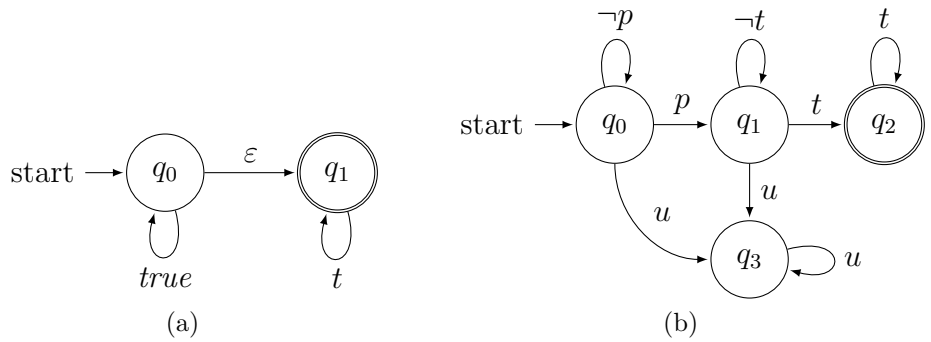


Figure 4.4: (a) LDBA for the specification in (4.5) - (b) LDBA for property in (4.6).

to eventually find the target $\Diamond t$ and to stay there $\Box(t \rightarrow \Box t)$, while avoiding the unsafe - otherwise it is going to be trapped there $\Box(u \rightarrow \Box u)$. Specification (4.5) requires the agent to eventually find the target and to stay there. The intuition behind (4.6) is that the agent has to eventually first visit p and then visit t at some point in the future $\Diamond(p \wedge \Diamond t)$ and stay there $\Box(t \rightarrow \Box t)$ while avoiding unsafe areas $\Box(u \rightarrow \Box u)$.

The LDBAs associated with (4.4), (4.5) and (4.6) are in Fig. 4.3 and Fig. 4.4, respectively. Note that the LDBA expressing (4.4) in Fig. 4.3 is deterministic and only needs the state set $\mathcal{Q}_D$ as in Definition 2.10 while the LDBA in Fig. 4.4. a needs both $\mathcal{Q}_D$ and $\mathcal{Q}_N$ to express (4.5).

The second experiment is the well-known Atari 2600 game Pacman, which is initialised in a tricky configuration (Fig. 4.5). In order to win the game, the agent has to collect all available tokens without being caught by moving ghosts. The ghost dynamics is stochastic: a probability p_g for each ghost determines if the ghost is chasing Pacman (often referred to as "chase mode") or if it is executing a random action ("scatter mode"). Notice that, unlike the first experiment, in this setup, the actions of the ghosts and of the agent result in a deterministic transition, i.e. the world is not "slippery". Each combination of (Pacman, ghost1, ghost2, ghost3, ghost4) represents a state in the experiment, resulting in a state-space

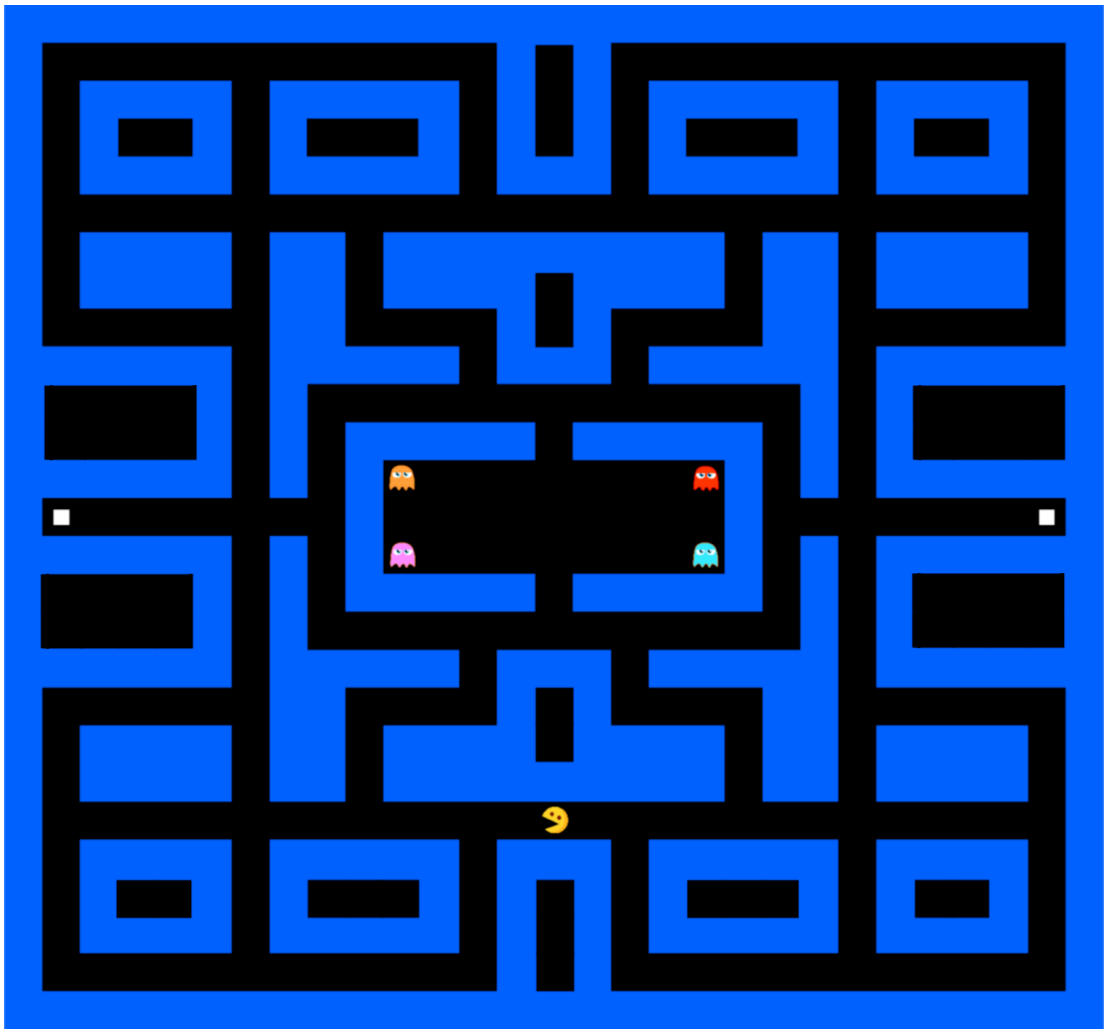


Figure 4.5: Pacman environment with $|\mathcal{S}| > 80{,}000$ – initial condition: the square on the left is labelled as food 1 (f_1) and the one on the right as food 2 (f_2), the state of being caught by a ghost is labelled as (g) and the rest of the state space is neutral (n).

cardinality in excess of 80,000.

In the second experiment, in order to win the game, Pacman is required to choose between one of the two available foods and then find the other one ($\Diamond[(f_1 \wedge \Diamond f_2) \vee (f_2 \wedge \Diamond f_1)]$) while avoiding ghosts ($\Box\neg g$). These clauses are what a human can perceive just by looking at the game screen and we feed the conjunction of these associations to the agent by using the following LTL formula:

$$\Diamond[(f_1 \wedge \Diamond f_2) \vee (f_2 \wedge \Diamond f_1)] \wedge \Box\neg g, \tag{4.7}$$

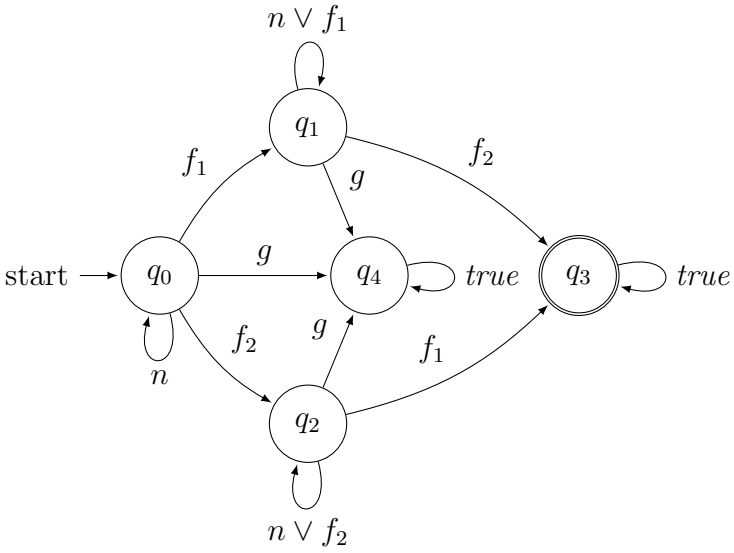


Figure 4.6: LDBA for the specification in (4.7).

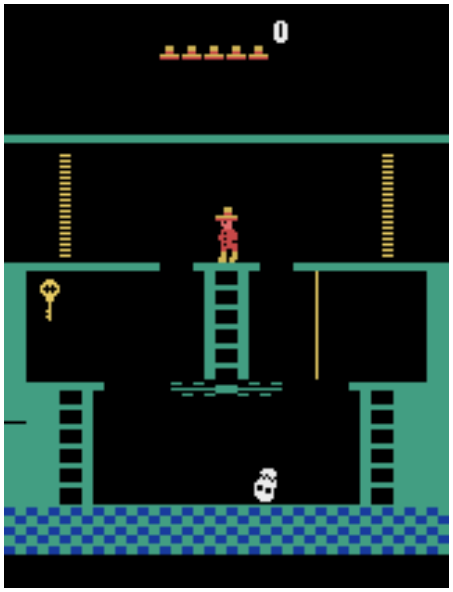


Figure 4.7: Initial condition in the first level of Montezuma's Revenge.

The constructed LDBA is shown in Fig. 4.6.

The third experiment deals with the complex environment of Atari 2600 Montezuma's Revenge. To win the game the agent needs to descend down the ladders, to jump over the skull, to fetch the key and to return to the top and to open one of the doors. In this experiment, we assume that for each action that the agent selects, there is a 90% that the chosen action is executed and a 10% that a

random action is instead performed.

Using the OpenAI gym environment [187] we set as our test-bed the first chamber of Montezuma's Revenge (Fig. 4.7). In order to enable a tabular approach, we reduce the size of the state-action space: This simplification is only a means of expediting the training process, but no generality is lost. The goal is to achieve a better score overall than some modern algorithms, such as the DQN approach, which on average has achieved a score of zero [15].

In the following, we describe the methodology of simplifying the environment to make the testing feasible. The game simulator is very general and therefore comes with 18 possible actions, many of which do not apply in our environment, e.g. "FIRE". We observe that only 6 actions, out of the 18 native ones, are active in the Montezuma's Revenge environment, whilst the remaining ones produce no change to the environment, so exploring them is a waste of exploration time. As such, the algorithm could work just as well with the full set of actions, except that it would require additional time. We extract and work with pixel matrix to identify and locate different elements of the state space. Hence, we treat the state space as being discrete with over 700,000 states.

Montezuma's Revenge is a rather more complicated environment, where the probability of winning the game by randomly exploring the state space is close to zero. However, a human player can derive the logical property behind the game by simply looking at the map and associating the acquiring of the key to the ability of opening the door $\Diamond(k \wedge \Diamond d)$ and staying at the door to win the first chamber $\Box(d \rightarrow \Box d)$, while avoiding the moving skull $\Box(s \rightarrow \Box s)$. Here k represent the key, d is the door, and s is the skull. We can then combine these constraints and express them as an LTL formula, such as:

$$\Diamond(k \wedge \Diamond d) \wedge \Box(d \rightarrow \Box d) \wedge \Box(s \rightarrow \Box s), \tag{4.8}$$

Note that this LTL formula is equivalent to (4.6) and therefore we employ the LDBA in Fig. 4.4, however with different labels. The LDBA built from the formula encompasses the safety and the goal of the agent, however, to deal with the moving skull we represent the location and direction of the skull as a Kripke structure, which allows the agent to learn Q-values that generate policies avoiding the danger.

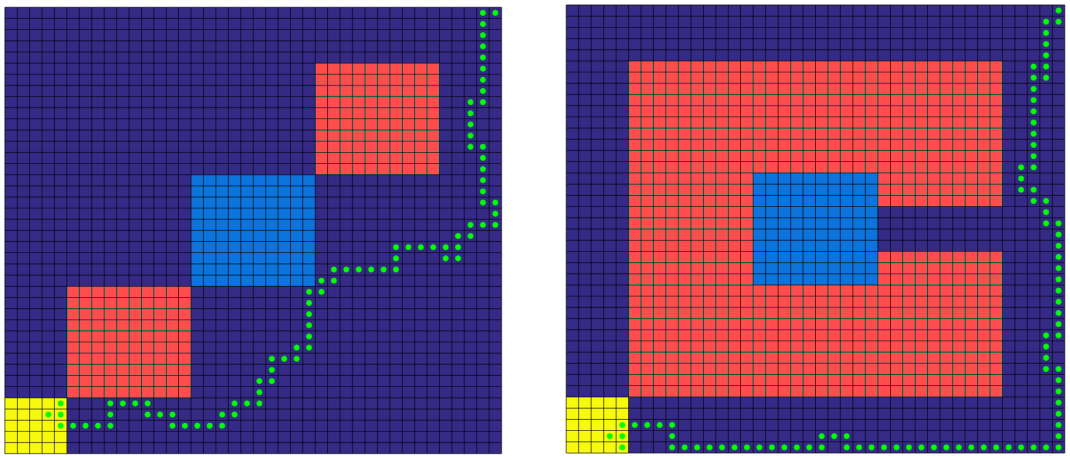
As mentioned earlier, the probability of randomly reaching the key and moving back to the door is very low, and even advanced algorithms, such as DQN [15], fail to achieve the overall goal. Of course, human intuition can solve this game and synthesise a successful policy – the advantage of our automated synthesis technique is that it does not require complete end-to-end examples, and thus it may avoid local optima that the human may believe to be global.

Note that the agent also loses a life when it falls from a high enough altitude, but this is not something that we can concretely assume and as such, it will not be penalised. Since we do not control the game engine, the agent will continue to lose its life upon falling, but rather than actively avoiding these moves, the agent will simply learn that the Q-value is null and there are better actions to be chosen.

4.3.1 Simulation Results

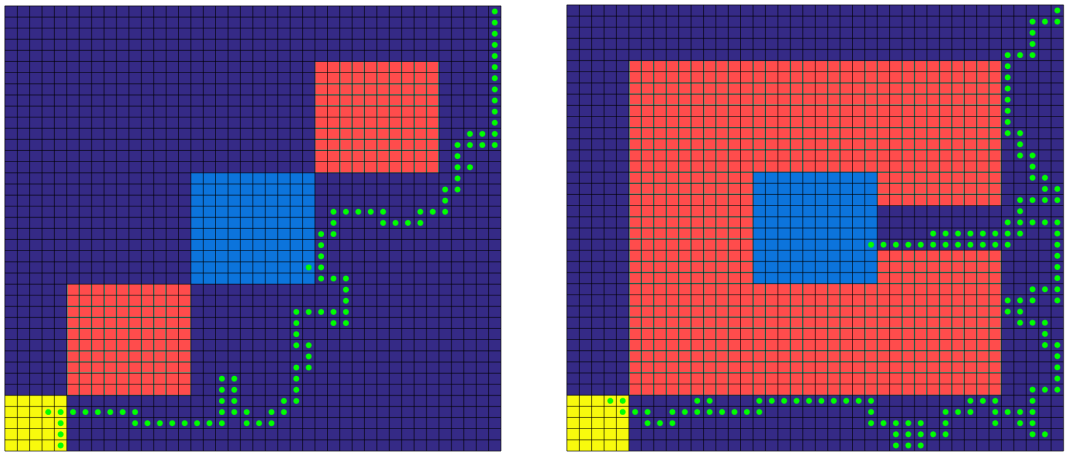
In the first experiment (slippery grid-world), the simulation parameters are set as $\mu = 0.9$ and $\gamma = 0.9$. Fig. 4.9 gives the results of the learning for the expression (4.6) in Region 1 and Region 2 after $400,000$ iterations and 200 learning episodes. Again, according to (4.6) the robot has to 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 selected action is executed with a probability of 85%. Thus, there might be some undesired deviations in the robot path.

Fig. 4.8 gives the results of the learning for the LTL formula (4.4) in Region 1 and Region 2 after $400,000$ iterations and 200 learning episodes. The intuition



(a) (b)

Figure 4.8: Simulation results for specification (4.4) in (a) Region 1 and in (b) Region 2.



(a) (b)

Figure 4.9: Simulation results for specification (4.6) in (a) Region 1 and (b) Region 2.

behind the LTL formula in (4.4) 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. 4.10 the learner tries to satisfy the LTL formula $\Diamond\Box t$ in (4.5). The learning takes 1000 iterations and 20 learning episodes.

Fig. 4.11 gives the result of our proposed value iteration method for calculating

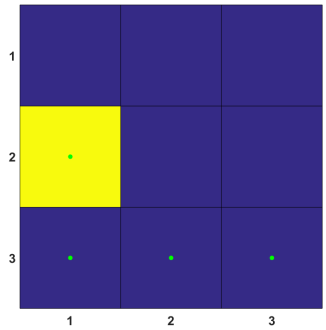
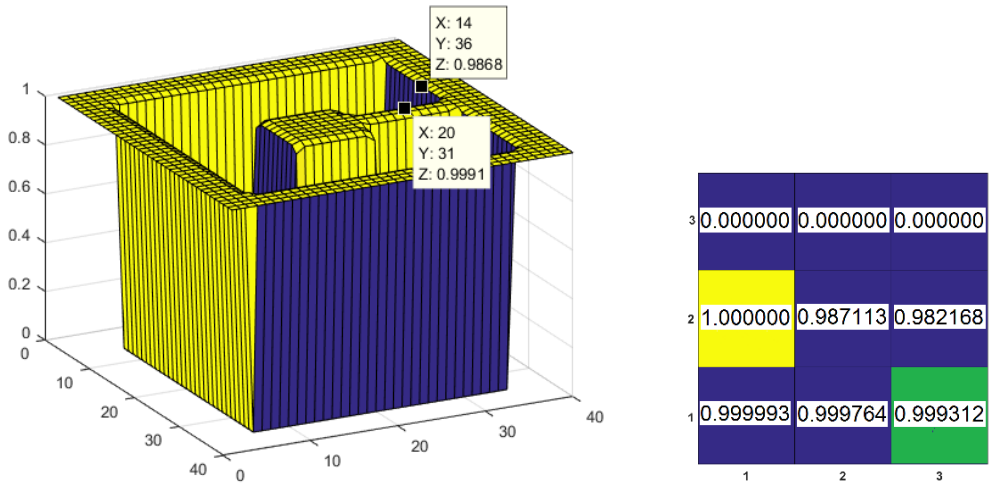


Figure 4.10: Simulation results for specification (4.5) in Region 3.

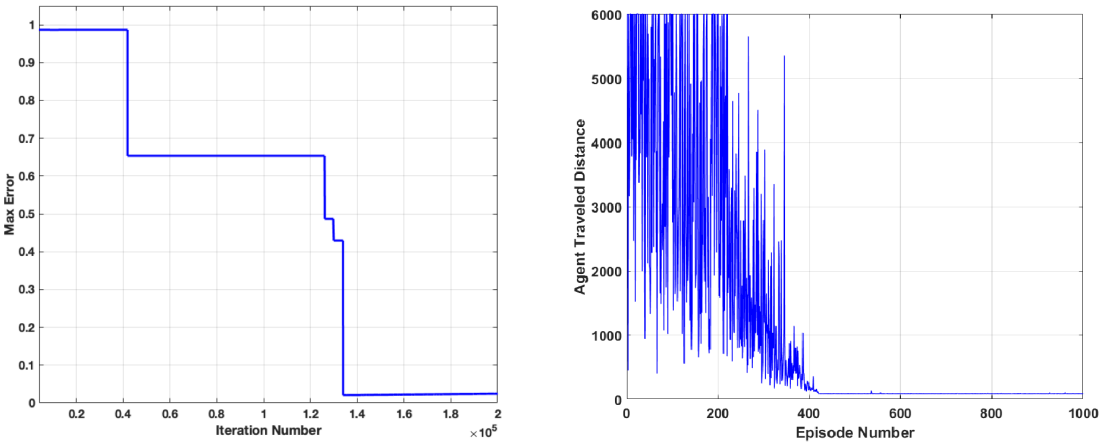


(a) (b)

Figure 4.11: PSP in (a) Region 2 with property (4.6) and in (b) Region 3 with specification (4.5). The results generated by LCQL are identical to the outcomes from PRISM.

the maximum PSP in Region 2 with (4.6) and Region 3 with (4.5). 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 LCQL (Fig. 4.12.a).

Fig. 4.12.b shows the distance that the agent traverses from initial state to final state at each learning episode in Region 1 under (4.6). After almost 400 episodes of learning the agent converges to the final optimal policy and the travelled distance stabilizes. It is worth mentioning that the application of standard QL has failed to



(a) (b)

Figure 4.12: In Region 2 under specification (4.6): (a) Maximum error between PSP computed with LCQL and with PRISM (b) The distance that agent traverses from initial state to final state with LCQL.

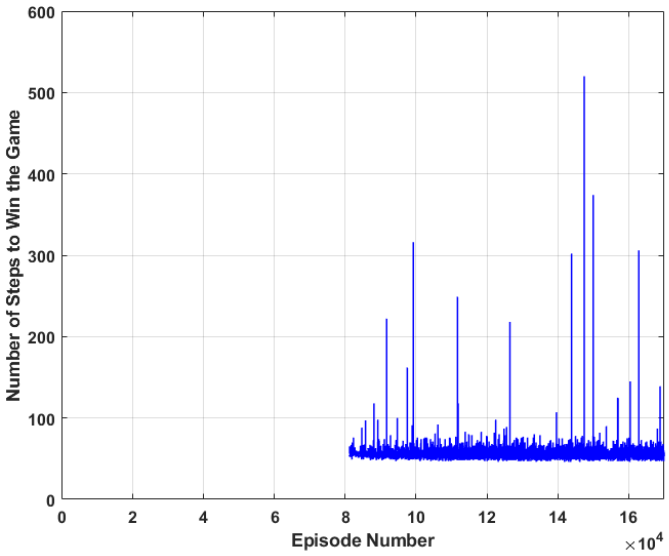


Figure 4.13: Results of learning in Pacman with LCQL (classical QL with positive reward for winning the game failed to converge and score even once).

find an optimal and stable policy for this experiment.

In the second experiment (Pacman), the simulation parameters are set as $\mu = 0.9$ and $\gamma = 0.9$. The stochastic behaviour of the ghosts is also captured by

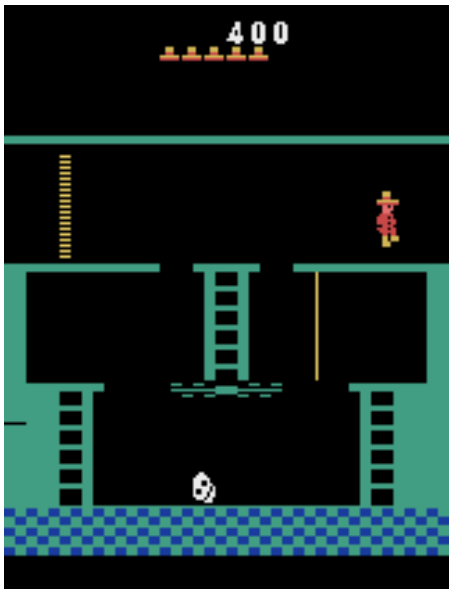


Figure 4.14: Montezuma's Revenge - The agent successfully unlocks the door (notice reward).

$p_g = 0.9$. Fig. 4.13 gives the results of learning with LCQL[1] for (4.6). After almost 80,000 episodes, LCQL finds a stable policy to win the game even with ghosts playing probabilistically. On the other hand, standard RL (in this case, classical QL with positive reward for winning the game) fails to find a stable policy.

In the third experiment (Montezuma's Revenge) the agent succeeds in reaching the set target after relatively short training (10000 episodes), which on a machine with a 3.2GHz Core i5 processor and 8GB of RAM, running Windows 7 took over a day, producing the results shown on Fig. 4.14. The simulation parameters are set as $\mu = 0.75$, $\gamma = 0.9$. For a DQN-based solution please refer to [5].

4.3.2 Comparison with a DRA-based Learning Algorithm

The problem of LTL-constrained learning is also investigated in [13], where the authors propose to translate the LTL property into a DRA and then to construct a product MDP. A 5×5 grid world is considered and starting from state $(0, 3)$ the

[1]Please visit https://grockious.github.io/lcrl to watch the videos of the agent playing Pacman. The code is adaptive and can be run under new configurations.

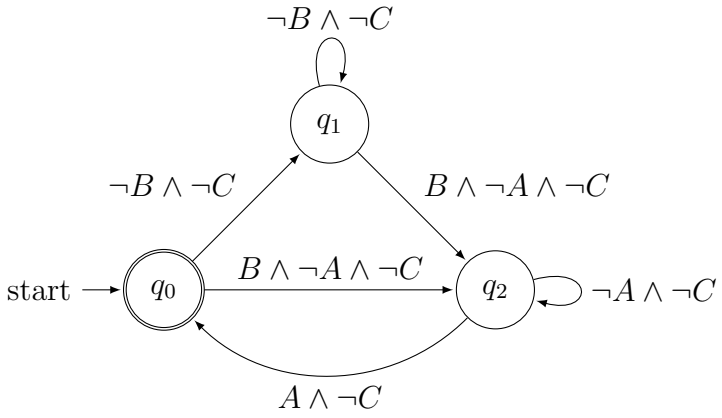


Figure 4.15: LDBA expressing the LTL formula in (4.9) with removed transitions labelled $A \wedge B$ (since it is impossible to be at A and B at the same time).

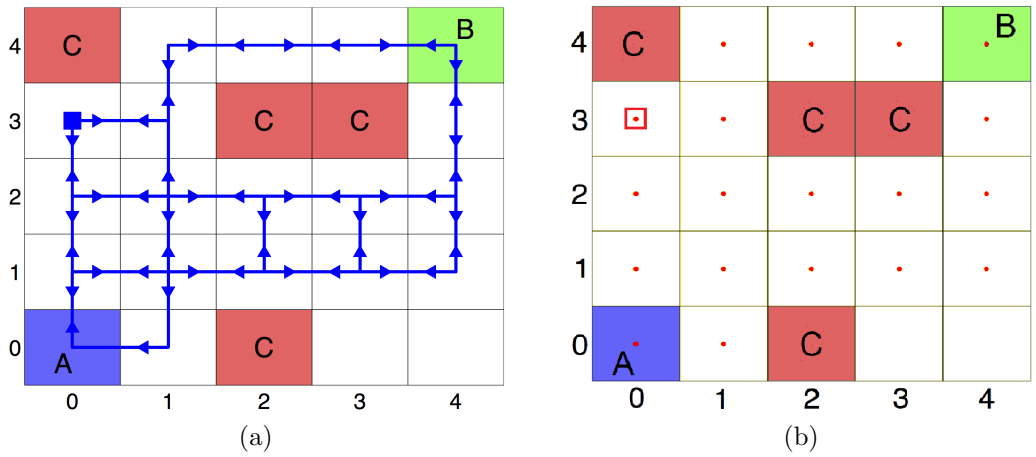


Figure 4.16: (a) Example considered in [13], and (b) Trajectories under the policy generated by LCQL in [13].

agent has to visit two regions infinitely often (areas A and B in Fig. 4.16). The agent has to also avoid the area C. This property can be encoded as the following LTL formula:

$$\square\Diamond A \wedge \square\Diamond B \wedge \square\neg C. \tag{4.9}$$

The product MDP in [13] contains 150 states, which means that the Rabin automaton has 6 states. Fig. 4.16.a shows the trajectories under the optimal policy generated by [13] algorithm after 600 iterations. However, by employing LCQL we

are able to generate the same trajectories with only 50 iterations (Fig. 4.16.b). The automaton that we consider is an LDBA with only 3 states as in Fig. 4.15. This result in a smaller product MDP and a much more succinct state space (only 75 states) for the algorithm to learn, which consequently leads to faster convergence.

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

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

Therefore for each $i \in \{1, \ldots, n_F\}$ a separate reward assignment is needed in [13] which complicates the implementation and increases the required calculation costs. This complicated reward assignment is not needed by employing the accepting frontier function in our scheme.

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

4.4 Summary

We have proposed LCQL, a model-free RL method to constrain the resulting traces by using an LTL property. The proposed algorithm is the first work on constraining model-free RL by an LTL specification which can be a foundation for future work in this direction. We have argued that converting the LTL property to a

GBA-based LDBA results in a significantly smaller automaton than to a DRA (the standard approach in the literature), which decreases the size of the product MDP and increases RL convergence rate. A DRA needs more complicated accepting conditions than a Büchi automaton; thus, a more complex reward assignment is required. 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 have proposed a value iteration method to calculate the probability of satisfaction of the LTL property in parallel with LCQL. We argue that with this trajectory-based method we are able to extend the satisfaction probability calculation to large-scale MDPs which are hard for model-based methods (such as DP) to handle. The 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.

The efficiency of DP is hindered by excessive memory requirements, caused by the need to store a full-model in memory and to apply the Bellman operator over the entire state-space. Thus, the main reason that LCQL improves performance and scalability is by avoiding an exhaustive update over the whole state space. In other words, LCQL "guides" the MDP exploration so as to minimize the solution time by only considering the portion of the MDP that is relevant to the LTL property. This is, in particular, made clear in the Pacman experiment, where we show that classical RL is very slow and ineffective in finding the optimal policy, whereas LCQL converges extremely fast thanks to the guidance that is provided by the automaton.

Chapter 5

Logically-Constrained Neural Fitted Q-iteration

5.1 Introduction

As shown in the last chapter LCQL is a promising paradigm for training an autonomous agent under LTL to make optimal decisions when interacting with a finite-state MDP if the stochastic behaviour of the MDP is initially unknown. However, LCQL is focused on problems in which the set of states of the MDP and the set of possible actions are both finite. Nonetheless, many interesting real-world problems require actions to be taken in response to high-dimensional or real-valued sensory inputs [48]. As an example, consider the problem of drone control in which the drone state is represented as its Euclidean position $(x, y, z) \in \mathbb{R}^3$: the state space of an MDP modelling the stochastic behaviour of the drone is uncountably infinite, namely continuous.

We can always apply LCQL to an infinite-state MDP by discretising the state space of the MDP a-priori in order to find the optimal policy [41]. Although this method works well for many problems, the resulting discrete MDP is often inaccurate and may not capture the full dynamics of the original MDP. Thus, as discussed before, the discretisation of MDPs suffers from the trade-off between accuracy and the curse of dimensionality.

Alternatively, we can collect a number of samples and only then apply an approximation function that is constructed via regression over the set of samples. The approximation function essentially replaces the conventional LCQL state-action-reward look-up table by generalising over the state space of the MDP.

In this chapter, we extend LCRL architecture to the first model-free RL algorithm, based on Neural Fitted Q-iteration (NFQ), that can synthesise a policy satisfying a given LTL property when the given MDP has a continuous state space. We call this algorithm Logically-Constrained NFQ (LCNFQ) and we show that the proposed architecture is efficient and is compatible with RL algorithms that are the core of recent developments in the community. The experiments showcase the results and efficiency of LCNFQ with respect to other approaches in training time, sample complexity and success rate.

5.2 Logically-Constrained NFQ

Recall from Section 2.2.2 that NFQ employs feedforward neural networks [168] to approximate the Q-function, namely to efficiently generalise or interpolate it over the entire state space, exploiting a finite set of experience samples. This set is called experience replay. Instead of the conventional QL update rule in (2.4), NFQ introduces a loss function that measures the error between the current Q-values $Q(s,a)$ and their target value $R(s,a) + \gamma \max_{a'} Q(s',a')$, namely

$$L = [Q(s,a) - R(s,a) + \gamma \max_{a'} Q(s',a')]^2. \tag{5.1}$$

In LCNFQ, the experience replay method is adapted to the product MDP structure, over which we let the agent explore the MDP and reinitialise it when a positive reward is received or when no positive reward is received after a given number *th* of iterations. The parameter *th* is set manually according to the state space of the MDP, allowing the agent to explore the MDP while keeping the size of the sample set limited. All the traces that are gathered within episodes, i.e. experiences, are stored in the form of $(s^{\otimes}, a, s^{\otimes\prime}, R(s^{\otimes}, a), q)$, where $s^{\otimes} = (s, q)$ is the current state in the product MDP, a is the selected action, $s^{\otimes\prime} = (s', q')$ is the subsequent state, and $R(s^{\otimes}, a)$ is the reward gained as in (3.1). The set of past experiences is called the sample set $\mathcal{E}$.

Once the exploration phase is completed and the sample set is created, learning is performed over the sample set. In the learning phase, we propose a hybrid architecture of n separate feedforward neural nets, each with one hidden layer, where $n = |\mathcal{Q}|$ and $\mathcal{Q}$ is the finite cardinality of the automaton $\mathfrak{A}$[1]. Each neural net is associated with a state in the LDBA and for each automaton state $q_i \in \mathcal{Q}$ the associated neural net is called $B_{q_i} : \mathcal{S}^{\otimes} \times \mathcal{A} \to \mathbb{R}$. Once the agent is at state $s^{\otimes} = (s, q_i)$ the neural net B_{q_i} is used for the local Q-function approximation. The set of neural nets acts as a global hybrid Q-function approximator $Q : \mathcal{S}^{\otimes} \times \mathcal{A} \to \mathbb{R}$. Note that the neural nets are not fully decoupled. For example, assume that by taking action a in state $s^{\otimes} = (s, q_i)$ the agent is moved to state $s^{\otimes\prime} = (s', q_j)$ where

[1]Different embeddings, such as the one hot encoding [188] and the integer encoding, have been applied in order to approximate the global Q-function with a single feedforward net. However, we have observed poor performance since these encodings allow the network to assume an ordinal relationship between automaton states. This means that by assigning integer numbers or one hot codes, automaton states are categorised in an ordered format, and can be ranked. Clearly, this disrupts Q-function generalisation by assuming that some states in product MDP are closer to each other. Consequently, we have turned to the use of n separate neural nets, which work together in a hybrid fashion, meaning that the agent can switch between these neural nets as it jumps from one automaton state to another.

Algorithm 5.1: Logically-Constrained NFQ

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

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

$q_i \neq q_j$. According to (2.6) the weights of B_{q_i} are updated such that $B_{q_i}(s^{\otimes}, a)$ has minimum possible error to $R(s^{\otimes}, a) + \gamma \max_{a'} B_{q_j}(s^{\otimes\prime}, a')$. Therefore, the value of $B_{q_j}(s^{\otimes\prime}, a')$ affects $B_{q_i}(s^{\otimes}, a)$.

Let $q_i \in \mathcal{Q}$ be a state in the LDBA. Then define $\mathcal{E}_{q_i} := \{(\cdot, \cdot, \cdot, \cdot, x) \in \mathcal{E} | x = q_i\}$ as the set of experiences within $\mathcal{E}$ that are associated to state q_i, i.e. $\mathcal{E}_{q_i}$ is the projection of $\mathcal{E}$ onto q_i. Once the exploration phase is completed, each neural net B_{q_i} is trained based on the associated experience set $\mathcal{E}_{q_i}$. At each iteration of training, a pattern set $\mathcal{P}_{q_i}$ is generated based on the experience set $\mathcal{E}_{q_i}$:

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

where $input_l = ({s_l}^{\otimes}, a_l)$ and $target_l = R({s_l}^{\otimes}, a_l) + \gamma \max_{a'} Q({s_l}^{\otimes\prime}, a')$ such that $({s_l}^{\otimes}, a_l, {s_l}^{\otimes\prime}, R({s_l}^{\otimes}, a_l), q_i) \in \mathcal{E}_{q_i}$. In each epoch of LCNFQ (Algorithm 5.1), the pattern set $\mathcal{P}_{q_i}$ is used as the input-output set to train the neural net B_{q_i}. In order to update the weights in each neural net, we use Rprop [189] for its efficiency in batch learning [56]. The training schedule in the hybrid network starts from individual networks that are associated with accepting states of the automaton. The training sequence goes backward until it reaches the networks that are associated

to the initial states. By doing so, we allow the Q-value to back-propagate through the connected networks. In Algorithm 5.1, without loss of generality we assume that the automaton states are ordered and hence the back-propagation starts from $q_i = |\mathcal{Q}|$.

Note that proving that an algorithm that synthesises a policy maximising the associated expected discounted return over the product MDP, maximises the probability of satisfying the property is not trivial as it is in the case of finite MDPs. More specifically we cannot leverage notions that are specific to finite MDPs, such as that of Accepting Max End Component (AMEC). Thus, the probability of satisfaction can not be equated to the probability of reaching a set of states in the product MDP (i.e., the AMEC) and we have to directly reason over the accepting condition of the LDBA.

Theorem 5.1. *Let φ be a given LTL formula and $\mathfrak{M}_\mathfrak{A}$ be the product MDP constructed by synchronising the MDP $\mathfrak{M}$ with the LDBA $\mathfrak{A}$ expressing φ. The optimal stationary Markov policy on $\mathfrak{M}_\mathfrak{A}$ that maximises the expected return, maximises the probability of satisfying φ and induces a finite-memory policy on the MDP $\mathfrak{M}$.*

Proof. Assume that the optimal Markov policy on $\mathfrak{M}_\mathfrak{A}$ is $\pi^{\otimes *}$, namely at each state $s^\otimes$ in $\mathfrak{M}_\mathfrak{A}$ we have

$$\pi^{\otimes *}(s^\otimes) = \underset{\pi^\otimes \in \mathcal{D}^\otimes}{\operatorname{argsup}} U^{\pi^\otimes}(s^\otimes) = \underset{\pi^\otimes \in \mathcal{D}^\otimes}{\operatorname{argsup}} \mathbb{E}^{\pi^\otimes}[\sum_{n=0}^{\infty} \gamma^n \; R(s_n^\otimes, a_n) | s_0^\otimes = s^\otimes], \tag{5.2}$$

where $\mathcal{D}^\otimes$ is the set of stationary deterministic policies over the state space $\mathcal{S}^\otimes$, $\mathbb{E}^{\pi^\otimes}[\cdot]$ denotes the expectation given that the agent follows policy $\pi^\otimes$, and $s_0^\otimes, a_0, s_1^\otimes, a_1, \ldots$ is a generic path generated by the product MDP under policy $\pi^\otimes$.

Recall that an infinite word $w \in \Sigma^\omega$, $\Sigma = 2^{\mathcal{A} \cup \mathcal{P}}$ is accepted by the LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ 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 $F_j \in \mathcal{F}$, $inf(\theta) \cap F_j \neq \emptyset$, where $inf(\theta)$ is the

set of states that are visited infinitely often in the sequence θ. From Definition 2.9, the associated run θ of an infinite path in the product MDP $\rho = s_0^{\otimes} \xrightarrow{a_0} s_1^{\otimes} \xrightarrow{a_1} ...$ is $\theta = L^{\otimes}(s_0^{\otimes})L^{\otimes}(s_1^{\otimes})....$

From Definition 3.2 and (3.1), and since for an accepting run $inf(\theta) \cap F_j \neq \emptyset,\ \forall F_j \in \mathcal{F}$, all accepting paths starting from $s_0^{\otimes}$, accumulate infinite number of positive rewards r_p (see Remark 3.2).

In the following, by contradiction, we show that any optimal policy $\pi^{\otimes *}$ satisfies the property with maximum possible probability. Let us assume that there exists a stationary deterministic Markov policy $\pi^{\otimes +} \neq \pi^{\otimes *}$ over the state space $\mathcal{S}^{\otimes}$ such that probability of satisfying φ under $\pi^{\otimes +}$ is maximum.

This essentially means in the product MDP $\mathfrak{M}_{\mathfrak{A}}$ by following $\pi^{\otimes +}$ the expectation of reaching the point where $inf(\theta) \cap F_j \neq \emptyset,\ \forall F_j \in \mathcal{F}$ and positive reward is received ever after is higher than any other policy, including $\pi^{\otimes *}$. With a tuned discount factor γ, e.g. (2.2),

$$\mathbb{E}^{\pi^{\otimes +}}[\sum_{n=0}^{\infty} \gamma^n\ R(s_n^{\otimes}, a_n) | s_0^{\otimes} = s^{\otimes}] > \mathbb{E}^{\pi^{\otimes *}}[\sum_{n=0}^{\infty} \gamma^n\ R(s_n^{\otimes}, a_n) | s_0^{\otimes} = s^{\otimes}]. \tag{5.3}$$

This is in contrast with optimality of $\pi^{\otimes *}$ (5.2) and concludes $\pi^{\otimes *} = \pi^{\otimes +}$. Namely, an optimal policy that maximises the expected return also maximises the probability of satisfying LTL property φ. It is easy to see that the projection of policy $\pi^{\otimes *}$ on MDP $\mathfrak{M}$ is a finite-memory policy π^*. □

Remark 5.1. *The optimality of the policy generated by LCNFQ also depends on other factors, such as the neural network structure, the number of hidden layers, and the choice of activation functions. The quantification of the sub-optimality of the policy generated due to these factors is out of the scope of this work.*

5.3 Alternatives to LCNFQ

In the following, we discuss the most popular alternative approaches to solving infinite-state MDPs, namely the Voronoi Quantiser (VQ) and Fitted Value Iteration (FVI). They will be benchmarked against LCNFQ in later sections.

5.3.1 Voronoi Quantiser (VQ)

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

Inspired by [109], we propose a version of VQ that is able to discretise the state space of the product MDP $\mathcal{S}^{\otimes}$, while allowing RL to explore the MDP. VQ maps the state space onto a finite set of disjoint regions called Voronoi cells [154]. The set of centroids of these cells is denoted by

$$\mathcal{C} = \{c_i\}_{i=1}^{m},\ c_i \in \mathcal{S}^{\otimes},$$

where m is the number of the cells. With $\mathcal{C}$, we are able to use QL and find an approximation of the optimal policy for a continuous-state MDP.

In the beginning, $\mathcal{C}$ is initialised to consist of just one c_1, which corresponds to the initial state. This means that the agent views the entire state space as a homogeneous region when no a-priori knowledge is available. Assuming that states are represented by vectors, when the agent explores this unknown state space, the

Algorithm 5.2: Episodic VQ

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

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

Euclidean norm of the distance between each newly visited state and its nearest neighbour can be calculated. If this norm is greater than a threshold value Δ called "minimum resolution", or if the new state $s^{\otimes}$ comprises an automaton state that has never been visited, then the newly visited state is appended to $\mathcal{C}$. Therefore, as the agent continues to explore, the size of $\mathcal{C}$ would increase until the "relevant" parts of the state space are partitioned. In our algorithm, the set $\mathcal{C}$ has $|\mathcal{Q}|$ disjoint subsets where $\mathcal{Q}$ is the finite set of states of the automaton. Each subset $\mathcal{C}^{q_j},\ j = 1, ..., |\mathcal{Q}|$ contains the centroids of those Voronoi cells that have the form of $c_i^{q_j} = (\cdot, q_j)$, i.e.

$\bigcup_i^m c_i^{q_j} = \mathcal{C}^{q_j}$ and $\mathcal{C} = \bigcup_{j=1}^{|\mathcal{Q}|} \mathcal{C}^{q_j}$. Therefore, a Voronoi cell

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

is defined by the nearest neighbour rule for any $i' \neq i$. The proposed VQ algorithm is presented in Algorithm 5.2.

5.3.2 Fitted Value Iteration (FVI)

FVI is an approximate DP algorithm for continuous-state MDPs, which employs function approximation techniques [110]. In standard DP the goal is to find a mapping, i.e. value function, from the state space to $\mathbb{R}$, which can generate the optimal policy. The value function in our setup is the expected return in (2.1) when π is the optimal policy, i.e. U^{π^*}. Over continuous state spaces, analytical representations of the value function are in general not available. Approximations can be obtained numerically through approximate value iterations, which involve approximately iterating a Bellman operator on an approximate value function [41].

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

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

$$\tau v^{q_j}(s) = \sup_{a \in \mathcal{A}} \{ \int v(y) P^{\otimes}(dy|(s, q_j), a) \}, \tag{5.4}$$

where τ is the Bellman operator [190]. The update in (5.4) is different from the

standard Bellman update in DP, as it does not comprise a running reward, and as the (terminal) reward is replaced by the following function initialization:

$$v(s^{\otimes}) = \begin{cases} r_p & \text{if } s^{\otimes} \in \mathbb{A}, \\ r_n & \text{otherwise.} \end{cases} \tag{5.5}$$

Here r_p and r_n are defined in (3.1) with $y = 0$. The main hurdle in executing the Bellman operator in continuous state MDPs, as in (5.4), is that no analytical representations of the value function v and of its components $v^{q_j},\ q_j \in \mathcal{Q}$ are in general available. Therefore, we employ an approximation method, by introducing a new operator L.

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

In the proposed FVI algorithm, we employ the kernel averager method [41], which can be represented by the following expression for each state (s, q_j):

$$Lv(s, q_j) = Lv^{q_j}(s) = \frac{\sum_{i=1}^{k} K(s_i - s) v^{q_j}(s_i)}{\sum_{i=1}^{k} K(s_i - s)}, \tag{5.6}$$

where the kernel $K : \mathcal{S} \to \mathbb{R}$ is a radial basis function, such as $e^{-|s-s_i|/h'}$, and h' is smoothing parameter. Each kernel is characterised by the point s_i, and its value decays to zero as s diverges from s_i. This means that for each $q_j \in \mathcal{Q}$ the approximation operator L in (5.6) is a convex combination of the values of the centres $\{s_i\}_{i=1}^{k}$ with larger weight given to those values $v^{q_j}(s_i)$ for which s_i is close to s. Note that the smoothing parameter h' controls the weight assigned to more distant values.

In order to approximate the integral in the Bellman update (5.4) we use a

Algorithm 5.3: FVI

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

output : approximated value function Lv

initialize Lv

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

repeat

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

    $\forall q_j \in \mathcal{Q},\ \forall i = 1, ..., k\ ,\ \forall a \in \mathcal{A}$ calculate $I_a((s_i, q_j)) = 1/Z \sum_{y \in \mathcal{Y}_a^Z(s_i, q_j)} Lv(y)$ using (5.6)

    for each state (s_i, q_j), update $v^{q_j}(s_i) = \sup_{a \in \mathcal{A}} \{I_a((s_i, q_j))\}$ in (5.6)

  end

until *end of trial*

Monte Carlo sampling technique [191]. For each centre (s_i, q_j) and for each action a, we sample the next state $y_a^z(s_i, q_j)$ for $z = 1, ..., Z$ times and append these samples to the set of Z subsequent states $\mathcal{Y}_a^Z(s_i, q_j)$. We then replace the integral with

$$I_a(s_i, q_j) = \frac{1}{Z} \sum_{z=1}^{Z} Lv(y_a^z(s_i, q_j)). \tag{5.7}$$

The approximate value function Lv is initialised according to (5.5). In each loop in FVI, the Bellman update approximation is first executed over those sub-value functions that are linked with the accepting states of the LDBA, i.e. those that have an initial value of r_p. The approximate Bellman update then back-propagates this value until it reaches those sub-value functions that are linked with the initial states of the automaton. This allows the state values to back-propagate through the product MDP transitions that connect the sub-value function via (5.7). Without loss of generality, we assume that the automaton states are ordered and hence the back-propagation starts from $q_i = |\mathcal{Q}|$. Once we have the approximated value function, we can generate the optimal policy by following the maximum value (Algorithm 5.3).

We conclude this section by emphasising that Algorithms 5.2 and 5.3 are

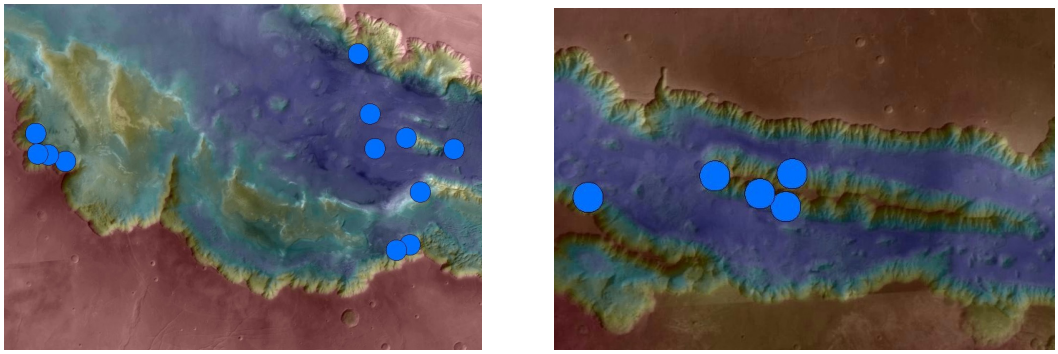
proposed to be benchmarked against LCNFQ. Further, it is worth mentioning that MDP abstraction techniques such as [38] failed to scale and to find an optimal policy.

5.4 Experimental Results

In the following, we describe a mission planning architecture for an autonomous Mars rover that uses LCNFQ to follow a mission on Mars. We start with a satellite image from the surface of Mars, and add the desired labels from $2^{\mathcal{AP}}$, e.g. safe or unsafe, to that image. Next, we express the desired mission task by an LTL formula defined over $\mathcal{AP}$, and then we run LCNFQ on the labelled image. We would like the rover to satisfy the given LTL property with the highest probability possible starting from any random initial state (as we assume we cannot predict the landing location precisely). We compare LCNFQ with Voronoi quantizer and FVI and we show that LCNFQ outperforms these methods.

In this numerical experiment, we focus on the Coprates quadrangle in which there exist a significant number of signs of water. We consider two parts of Valles Marineris, a canyon system in the Coprates quadrangle (Fig. 5.1). The blue dots, provided by NASA, indicate locations of Recurring Slope Lineae (RSL) in the canyon network. RSL are seasonal dark streaks regarded as the strongest evidence for the possibility of liquid water on the surface of Mars. RSL extend down-slope during a warm season and then disappear in the colder part of the Martian year [192]. The two areas mapped in Fig. 5.1, Melas Chasma and Coprates Chasma, have the highest density of known RSL.

For each case, let the entire area be our MDP state space $\mathcal{S}$, where the rover location is a single state $s \in \mathcal{S}$. At each state $s \in \mathcal{S}$, the rover has a set of actions $\mathcal{A} = \{\mathit{left}, \mathit{right}, \mathit{up}, \mathit{down}, \mathit{stay}\}$ by which it is able to move to other states: at



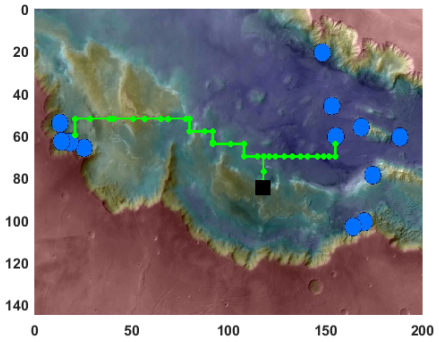
(a) Melas Chasma (b) Coprates Chasma

Figure 5.1: Melas Chasma and Coprates Chasma, in the central and eastern portions of Valles Marineris. Map colour spectrum represents elevation, where red is high and blue is low (Image courtesy of NASA, JPL, Caltech and University of Arizona).

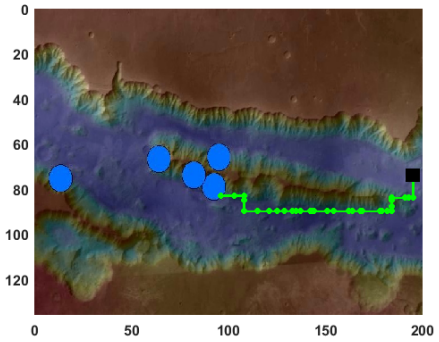
each state $s \in \mathcal{S}$, when the rover takes an action $a \in \{left, right, up, down\}$ it is moved to another state (e.g., s') towards the direction of the action with a range of movement that is randomly drawn from $(0, D]$ unless the rover hits the boundary of the area which forces the rover to remain on the boundary. In the case when the rover chooses action $a = stay$ it is again moved to a random place within a circle centred at its current state and with radius $d \ll D$. Again, d captures disturbances on the surface of Mars and can be tuned accordingly.

With $\mathcal{S}$ and $\mathcal{A}$ defined we are only left with the labelling function $L : \mathcal{S} \to 2^{\mathcal{AP}}$ which assigns to each state $s \in \mathcal{S}$ a set of atomic propositions $L(s) \subseteq 2^{\mathcal{AP}}$. With the labelling function, we are able to divide the area into different regions and define a logical property over the traces that the agent generates. In this particular experiment, we divide areas into three main regions: neutral, unsafe and target. The target label goes on RSL (blue dots), the unsafe label lays on the parts with very high elevation (red coloured), and the rest is neutral. In this example, we assume that the labels do not overlap each other.

Note that when the rover is deployed to its real mission, the precise landing location is not known. Therefore, we should take into account the randomness

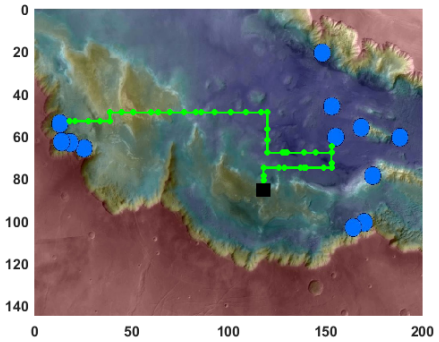


(a) Melas Chasma and landing location (black rectangle) (118, 85)

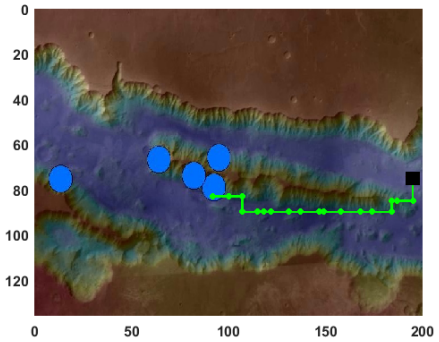


(b) Coprates Chasma and landing location (black rectangle) (194, 74)

Figure 5.2: Generated paths by LCNFQ.



(a) Melas Chasma and landing location (black rectangle) (118, 85)



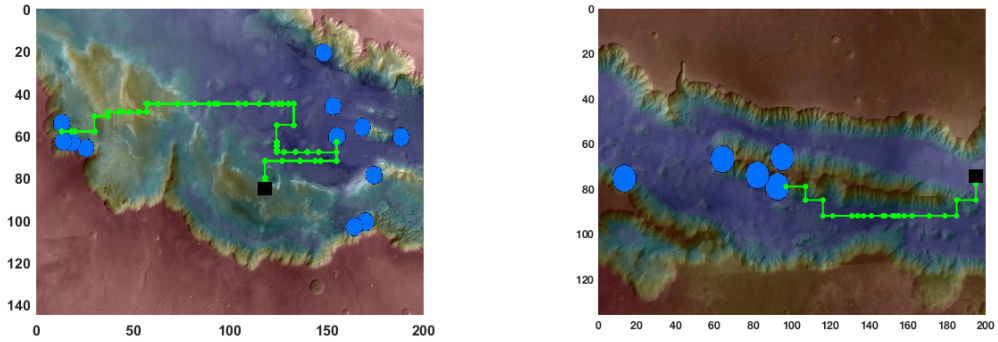
(b) Coprates Chasma and landing location (black rectangle) (194, 74)

Figure 5.3: Generated paths by episodic VQ.

of the initial state s_0. The dimensions of the area of interest in Fig. 5.1.a are 456.98×322.58 km and in Fig. 5.1.b are 323.47×215.05 km. The diameter of each RSL is 19.12 km. Other parameters in this numerical example have been set as $D = 2$ km, $d = 0.02$ km, the reward function parameter $y = 1$ for LCNFQ and $y = 0$ for VQ and FVI, $M = 1$, $m = 0.05$ and $\mathcal{AP} = \{$neutral, unsafe, target_1, target_2$\}$.

The first control objective in this numerical example is expressed by the following LTL formula over Melas Chasma (Fig. 5.1.a):

$$\Diamond(p \wedge \Diamond t) \wedge \Box(t \rightarrow \Box t) \wedge \Box(u \rightarrow \Box u), \tag{5.8}$$



(a) Melas Chasma and landing location (black rectangle) (118, 85)

(b) Coprates Chasma and landing location (black rectangle) (194, 74)

Figure 5.4: Generated paths by FVI.

where n stands for "neutral", p stands for "target 1", t stands for "target 2" and u stands for "unsafe". Target 1 is the RSL (blue bots) on the right with a lower risk of the rover going to an unsafe region, and the target 2 label is grounded on the left RSL that are a bit riskier to explore. Conforming to (5.8) the rover has to visit target 1 (any of the right dots) at least once and then proceed to target 2 (left dots) while avoiding unsafe areas. Note that according to $\Box(u \rightarrow \Box u)$ in (5.8) the agent is able to go to unsafe area u (by climbing up the slope) but it is not able to come back due to the risk of falling. Note that the LDBA expressing (5.8) is as in Fig. 4.4.a.

The second formula focuses more on safety and we are going to employ it in exploring Coprates Chasma (Fig. 5.1.b), where a critical unsafe slope exists in the middle of this region:

$$\Diamond t \wedge \Box(t \rightarrow \Box t) \wedge \Box(u \rightarrow \Box u) \tag{5.9}$$

Here, t refers to the "target", i.e. RSL in the map, and u stands for "unsafe". According to this LTL formula, the agent has to eventually reach the target ($\Diamond t$) and stays there ($\Box(t \rightarrow \Box t)$). However, if the agent hits the unsafe area it can never come back and remains there forever ($\Box(u \rightarrow \Box u)$). With (5.9) we can

Table 5.1: Simulation results for continuous-state MDPs.

Melas Chasma					
alg:lcql	**Sample Complexity**	$U^{\pi^*}(s_0)$	**Success Rate**†	**Training Time*****(s)**	**Iteration Num.**
LCNFQ	**7168** samples	**0.0203**	**99%**	95.64	**40**
VQ ($\Delta = 0.4$)	27886 samples	0.0015	**99%**	1732.35	2195
VQ ($\Delta = 1.2$)	7996 samples	0.0104	97%	273.049	913
VQ ($\Delta = 2$)	-	0	0%	-	-
FVI	40000 samples	0.0133	98%	**4.12**	80
Coprates Chasma					
alg:lcql	**Sample Complexity**	$U^{\pi^*}(s_0)$	**Success Rate**†	**Training Time*****(s)**	**Iteration Num.**
LCNFQ	**2680 samples**	**0.1094**	**98%**	166.13	**40**
VQ ($\Delta = 0.4$)	8040 samples	0.0082	**98%**	3666.18	3870
VQ ($\Delta = 1.2$)	3140 samples	0.0562	96%	931.33	2778
VQ ($\Delta = 2$)	-	0	0%	-	-
FVI	25000 samples	0.0717	97%	**2.16**	80

† Testing the trained agent (for 100 trials) * Average for 10 trainings

again build the associated Büchi automaton as in Fig. 4.4.b. Having the Büchi automaton for each formula, we are able to use Definition 3.1 to build product MDPs and run LCNFQ on both.

5.4.1 Simulation Results

This section presents the simulation results. All simulations are carried on a machine with a 3.2GHz Core i5 processor and 8GB of RAM, running Windows 7. LCNFQ has four feedforward neural networks for (4.6) and three feedforward neural networks for (4.4), each associated with an automaton state in Fig. 4.4.a and Fig. 4.4.b. We assume that the rover lands on a random safe place and has to find its way to satisfy the given property in the face of uncertainty. The learning discount factor γ is also set to be equal to 0.9.

Fig. 5.2 gives the results of learning for LTL formulae (4.6) and (4.4). At each state $s^\otimes$, the robot picks an action that yields highest $Q(s^\otimes, \cdot)$ and by doing so the robot is able to generate a control policy $\pi^{\otimes *}$ over the state space $\mathcal{S}^\otimes$. The control

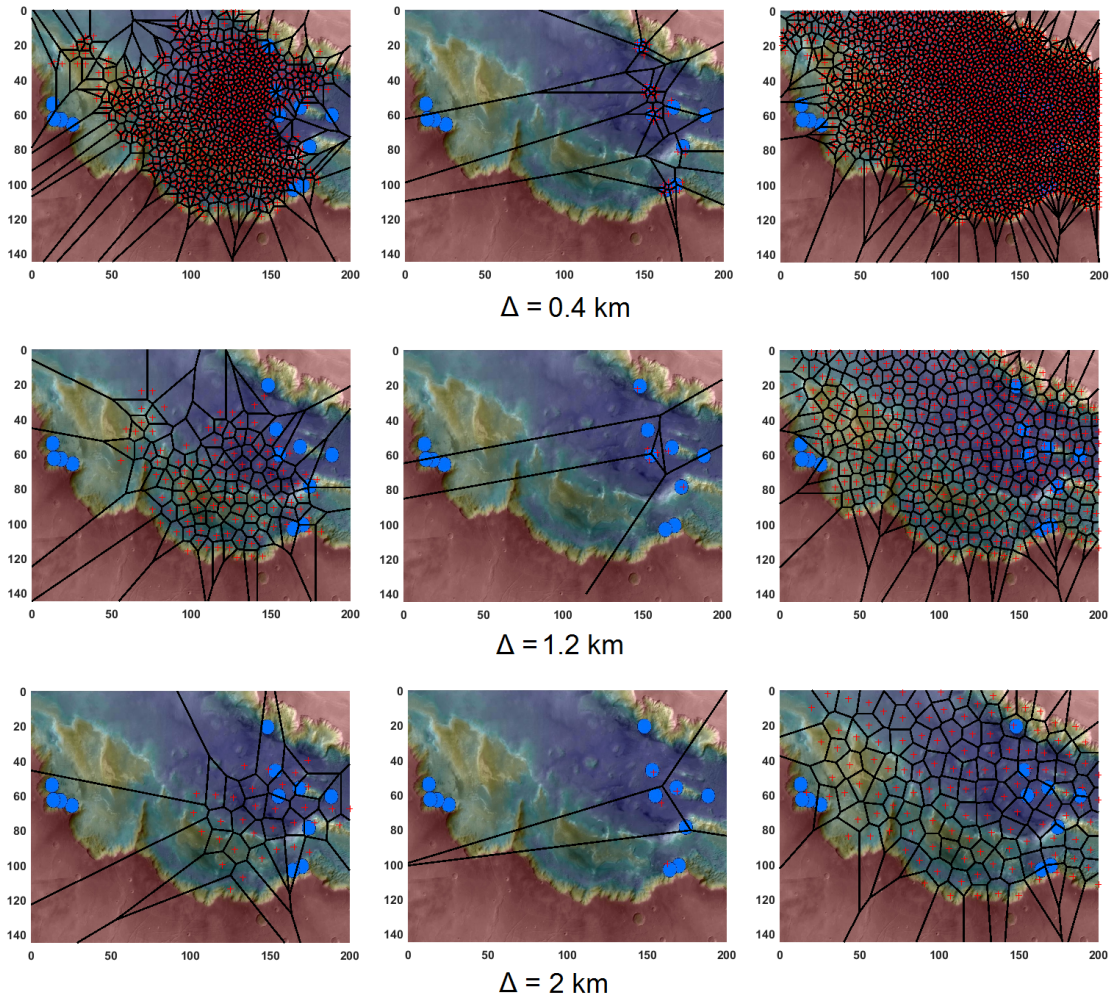


Figure 5.5: VQ – generated cells in Melas Chasma for different resolutions.

policy $\pi^{\otimes *}$ induces a policy π^* over the state space $\mathcal{S}$ and its performance is shown in Fig. 5.2.

Next, we investigate the episodic VQ algorithm as an alternative solution to LCNFQ. Three different resolutions ($\Delta = 0.4,\ 1.2,\ 2$ km) are used to see the effect of the resolution on the quality of the generated policy. The results are presented in Table 5.1, where VQ with $\Delta = 2$ km fails to find a satisfying policy in both regions, due to the coarseness of the resulted discretisation. A coarse partitioning results in the RL not being able to efficiently back-propagate the reward or the agent being stuck in some random-action loop as sometimes the agent current cell

is large enough that all actions have the same value. In Table 5.1, training time is the empirical time that is taken to train the algorithm and travel distance is the distance that the agent traverses from the initial state to the final state. We show the generated policy for $\Delta = 1.2$ km in Fig. 5.3. Additionally, Fig. 5.5 depicts the resulted Voronoi discretisation after implementing the VQ algorithm. Note that with VQ only those parts of the state space that are relevant to satisfying the property are accurately partitioned.

Finally, we present the results of FVI method in Fig. 5.4 for the LTL formulae (4.6) and (4.4). The FVI smoothing parameter is $h' = 0.18$ and the sampling time is $Z = 25$ for both regions where both are empirically adjusted to have the minimum possible value for FVI to generate satisfying policies. The number of basis points also is set to be 100, so the sample complexity of FVI is [2] equal to $100 \times Z \times |\mathcal{A}| \times (|\mathcal{Q}| - 1)$. Note that in Table 5.1, in terms of timing, FVI outperforms the other methods. However, we have to remember that FVI is an approximate DP algorithm, which inherently needs an approximation of the transition probabilities. Therefore, as we have seen in (5.7), for the set of basis points we need to perform Monte Carlo sampling for the subsequent states. This reduces the FVI applicability, as this sampling is not possible when dealing with a black-box model. In this numerical example, for the sake of comparison, we have assumed that FVI is able to perform Monte Carlo sampling so that it can be benchmarked against LCNFQ.

Additionally, both FVI and episodic VQ need careful hyper-parameter tuning to generate a satisfying policy, i.e., h' and Z for FVI and Δ for VQ. The big merit of LCNFQ is that it does not need any external intervention. Further, as in Table 5.1, LCNFQ succeeds to efficiently generate a better policy compared to FVI and VQ.

[2] We do not sample the states in the product automaton that are associated with the accepting state of the automaton since when we reach the accepting state the property is satisfied and there is no need for further exploration. Hence, the last term is $(|\mathcal{Q}| - 1)$. However, if the property of interest produces an automaton that has multiple accepting states, then we need to sample those states as well.

LCNFQ has less sample complexity while at the same time produces policies that are more reliable and also has better-expected return, i.e. higher probability of satisfying the given property.

5.5 Summary

This chapter proposed LCNFQ, the first RL algorithm to train Q-function in a continuous-state MDP such that the resulting traces satisfy an LTL property. LCNFQ employs multi-layer perceptrons to exploits the positive effects of generalization in neural networks, and uses separated experience replay sets to avoid the negative effects of disturbing previously learned experiences. This means that LCNFQ requires less experience and the learning process is highly data-efficient which subsequently increases the scalability and applicability of the proposed algorithm. LCNFQ is model-free, and thus the learning only depends on the sample experiences that the agent gathered by interacting and exploring the MDP. LCNFQ is successfully tested in numerical examples to verify its performance and it outperformed the competitors. Despite this success, LCNFQ is of limited applicability when the MDP action space is large or continuous. The next chapter takes a step further to improve the scalability of the LCRL architecture, and proposes a modular actor-critic deep learning that deals with the most general case of MDPs, i.e. continuous-state and continuous-action.

Chapter 6

Modular Deep Actor-critic Learning

6.1 Introduction

The previous chapter introduced LCNFQ, a model-free neural-fitted RL scheme and discussed its sample efficiency and performance. However, LCNFQ is an offline learning algorithm and is limited to problems where the action space is finite. In this chapter we propose an actor-critic, model-free, and online deep Reinforcement Learning (RL) scheme based on LCRL, for continuous-state continuous-action black-box environments when the reward is highly sparse but encompasses a high-level temporal structure represented as LTL, namely a formal, un-grounded, and symbolic representation of the task and its components.

Without requiring any supervision, each component of the LTL property systematically structures any complex mission task into low-level, achievable task

"modules". In other words, LTL acts as a guide for the agent within the product MDP, where a modular Deep Deterministic Policy Gradient (DDPG) [153] architecture is proposed to generate a low-level control policy. This synchronisation lifts deep RL applicability to complex temporal, non-Markovian, and memory-dependent policy synthesis problems.

Deep RL is an emerging paradigm for autonomous solving of decision-making tasks in complex and unknown environments. However, tasks featuring extremely delayed rewards are often difficult, if at all possible, to solve with monolithic learning in RL. In this work, we take a step back and recall that despite the generality of deep RL, this is not a natural representation of how humans perceive these problems, since humans already have prior knowledge and associations regarding elements and their corresponding function in a given environment.

These critical temporal high-level associations can lift deep RL initial knowledge about the problem to efficiently find the globally optimal policy, while avoiding an exhaustive unnecessary exploration, particularly early on. To the best of authors knowledge, no research has so far enabled RL to generate policies for general LTL properties over continuous-state continuous-action MDPs. Surely many real-world problems require continuous real-valued actions to be taken in response to high-dimensional and real-valued state observations.

A closely related work is curriculum learning [92], which masters easy instruction-based tasks first and eventually composes them together, to accomplish a more complex task. In this work instead, the complex task is expressed as an LTL property to guide learning and to directly generate policies, with no need to start from easier tasks and to later join corresponding policies together. The proposed method learns sub-tasks simultaneously in a one-shot learning scenario.

At the end of this chapter, we evaluate the performance of the proposed algorithm in continuous-state and -action Mars rover experiments, where we show

that

- LTL synchronisation lifts deep RL applicability to complex temporal, non-Markovian, and memory-dependent policy synthesis problems,
- we can skip curriculum learning steps and learn instead complex policies in one shot, and avoid an exhaustive exploration early on.

6.2 Modular DDPG

We consider an RL problem in which we exploit the structural information provided by the LTL specification and by constructing a sub-policy for each state of the associated LDBA. Our proposed approach learns a satisfying policy without requiring any information about the grounding of the LTL task to be explicitly specified. Namely, the labelling assignment in Definition 2.1 is unknown a-priori, and the algorithm solely relies on experience samples gathered on-the-fly.

Given an LTL mission task and an unknown black-box continuous-state and continuous-action MDP, we aim to synthesise a policy that satisfies the LTL specification. We relate the MDP and the automaton by synchronising them, in order to create a product MDP (Definition 3.1) that is first of all compatible with deep RL and secondly that encompasses the given logical property.

By constructing the product MDP we synchronise the current state of the MDP with the state of the automaton. This allows to evaluate the (partial) satisfaction of the corresponding LTL property (or parts thereof), and consequently to modularise the high-level task into sub-tasks. Hence, with a proper reward assignment driven from the LTL property and its associated LDBA, the agent is able to break down a complex task into a set of easy sub-tasks.

Before introducing the reward structure, let us recall the definition of non-accepting sink components of LDBA from Chapter 4:

Definition 6.1 (**Non-accepting Sink Component**). *A non-accepting sink component of the LDBA* $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ *is a directed graph induced by a set of states* $Q \subset \mathcal{Q}$ *such that (1) the graph is strongly connected; (2) it does not include all accepting sets* $F_k,\ k = 1, ..., f$*; and (3) there exist no other strongly connected set* $Q' \subset \mathcal{Q},\ Q' \neq Q$ *that* $Q \subset Q'$*. We denote the union of all non-accepting sink components of* $\mathfrak{A}$ *as* $\mathbb{N}$.

Based on the accepting frontier function in Definition 3.2, we propose a reward function that observes the current state $s^{\otimes}$, the current action a, and the subsequent state $s^{\otimes\prime}$, to provide the agent with a scalar value according to the current automaton state:

$$R(s^{\otimes}, a) = \begin{cases} r_p & if\ q' \in \mathbb{A},\ s^{\otimes\prime} = (s', q'), \\ r_n & if\ q' \in \mathbb{N},\ s^{\otimes\prime} = (s', q'), \\ 0 & otherwise. \end{cases} \tag{6.1}$$

Here r_p is a positive reward and r_n is a negative reward. A positive reward r_p is assigned to the agent when it takes an action that leads to a state, the label of which is in $\mathbb{A}$, and the reward is negative r_n upon reaching $\mathbb{N}$ (Definition 6.1). The set $\mathbb{A}$ is the accepting frontier set introduced in Section 3.3, and is initialised as the family set $\mathbb{A} = \{F_k\}_{k=1}^{f}$, and is updated by the following rule every time after the reward function is evaluated: $\mathbb{A} \leftarrow Acc(q', \mathbb{A})$.

Remark 6.1. *Recall that the set* $\mathbb{A}$ *contains those accepting states that are visited at a given time. Thus, the agent is guided by the above reward assignment to visit these states and once all of the sets* $F_k,\ k = 1, ..., f$*, are visited, the accepting frontier* $\mathbb{A}$ *is reset. As such, the agent is guided to visit the accepting sets infinitely often, and consequently, to satisfy the given LTL property.*

Given the product MDP structure in Definition 3.1 and the automatic formal reward assignment in (6.1), any algorithm that synthesises a policy maximising

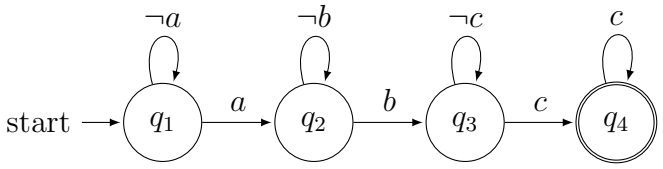


Figure 6.1: LDBA for a sequential mission task.

the associated expected discounted return over the product MDP, maximises the probability of satisfying the property φ. Note that, unlike the case of finite MDPs, proving the aforementioned claim is not trivial as we cannot leverage notions that are specific to finite MDPs, such as that of Accepting Max End Component (AMEC). Thus, the probability of satisfaction can not be equated to the probability of reaching a set of states in the product MDP (i.e., the AMEC) and we have to directly reason over the accepting condition of the LDBA.

Theorem 6.1. *Let φ be a given LTL formula and $\mathfrak{M}_\mathfrak{A}$ be the product MDP constructed by synchronising the MDP $\mathfrak{M}$ with the LDBA $\mathfrak{A}$ expressing φ. Then the optimal stationary Markov policy on $\mathfrak{M}_\mathfrak{A}$ that maximises the expected return, maximises the probability of satisfying φ and induces a finite-memory policy on the MDP $\mathfrak{M}$.*

Proof. The sketch of the proof is similar to the one of Theorem 5.1, and it relies on the contradiction one can draw if the optimal policy is not the policy that maximises the expected return. □

Remark 6.2. *Note that the optimality of the policy generated by the DDPG scheme depends on a number of factors, such as its structure, number of hidden layers, and activation functions. The quantification of the sub-optimality of the policy generated by DDPG is out of the scope of this work.*

6.2.1 Task Modularisation

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

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

The corresponding automaton for this LTL task is given in Fig. 6.1. The entire task is modularised into three sub-tasks, i.e. reaching a, b, and then c, and each automaton state acts as a divider.

Given an LTL task and its LDBA $\mathfrak{A} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$, we propose a modular architecture of $n = |\mathcal{Q}|$ separate actor, actor-target, critic and critic-target neural networks, along with their own replay buffer. A replay buffer is a finite-sized cache in which transitions sampled from exploring the environment are stored. The replay buffer is then used to train the actor and critic networks. The set of neural nets acts as a global modular actor-critic deep RL architecture, which allows the agent to jump from one sub-task to another by just switching between the set of neural nets. For each automaton state q_i an actor function $\mu_{q_i}(s|\theta^{\mu_{q_i}})$ represents the current policy by deterministically mapping states to actions, where $\theta^{\mu_{q_i}}$ is the vector of parameters of the function approximation for the actor. The critic $Q_{q_i}(s, a|\theta^{Q_{q_i}})$ is learned based on (2.7).

The modular DDPG algorithm is detailed in Algorithm 6.1. Each actor-critic network set in this algorithm is associated with its own replay buffer $\mathcal{R}_{q_i}$, where $q_i \in \mathcal{Q}$ (line 4, and line 12). Experience samples are stored in $\mathcal{R}_{q_i}$ in the form of

$$(s_i^{\otimes}, a_i, R_i, s_{i+1}^{\otimes}) = ((s_i, q_i), a_i, R_i, (s_{i+1}, q_{i+1})).$$

When the replay buffer reaches its maximum capacity, the samples are discarded based on a first in first out policy. At each time-step, actor and critic are updated by sampling a mini-batch of size $\mathcal{M}$ uniformly from $\mathcal{R}_{q_i}$. We only train the actor-critic network set corresponding to the current automaton state, as experience samples on the current automaton state have little influence on other actor-critic networks (line 12-17).

Further, directly implementing the update of the critic parameters as in (2.7) is shown to be potentially unstable, and as a result, the Q-update (line 14) is prone to divergence [193]. Hence, instead of directly copying the weights, the standard DDPG [153] uses "soft" target updates to improve learning stability. Target networks, Q' and μ', are time-delayed copies of the original actor and critic networks that slowly track the learned networks, Q and μ. These target actor and critic networks are used within the algorithm to gather evidence (line 13) and subsequently to update the actor and critic networks. In our algorithm, for each automaton state q_i we make a copy of the actor and the critic network: $\mu'_{q_i}(s|\theta^{\mu'_{q_i}})$ and $Q'_{q_i}(s,a|\theta^{Q'_{q_i}})$ respectively. The weights of both target networks are then updated by $\theta' = \tau\theta + (1-\tau)\theta'$ with a rate of $\tau < 1$ (line 18).

6.3 Experimental Results

In what follows, we discuss a mission planning problem for an autonomous Mars rover that uses the proposed algorithm to pursue exploration missions. The areas of interest on Mars are the Melas Chasma and the Victoria crater.

The Melas Chasma a number of signs of water, with ancient river valleys and networks of stream channels showing up as sinuous and meandering ridges and lakes (Fig. 6.2. a). The blue dots, provided by NASA, indicate locations of Recurring Slope Lineae (RSL) in the canyon network. RSL are seasonal dark

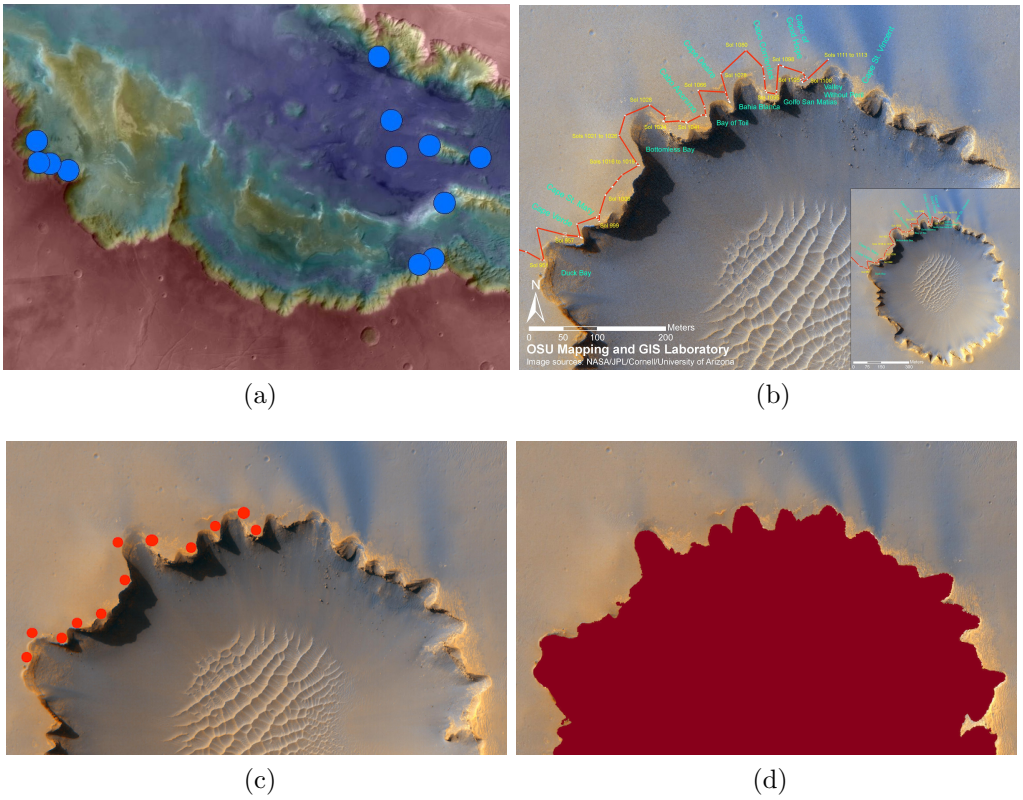
Algorithm 6.1: Modular DDPG

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

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

streaks regarded as the strongest evidence for the possibility of liquid water on the surface of Mars [192].

Victoria crater (Fig. 6.2. b) is an impact crater and is located near the equator of Mars. Layered sedimentary rocks are exposed along the wall of the crater, providing invaluable information about the ancient surface condition of Mars. Since January 2004, the well-known Mars rover Opportunity had been operating around the crater and its mission path is given in Fig. 6.2. b. Opportunity worked nearly 15 years on Mars and found dramatic evidence that long ago Mars was wetter and it could have sustained microbial life, if any existed.



(a) (b)

(c) (d)

Figure 6.2: (a) Melas Chasma in the Coprates quadrangle, map colour spectrum represents elevation, where red is high and blue is low. (b) Victoria crater and Opportunity rover mission traverse map, (c) replicated points of interest, and (d) unsafe area. (Image courtesy of NASA, JPL, Cornell, and Arizona University)

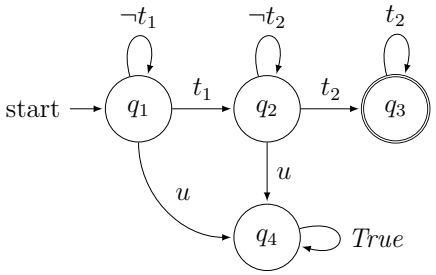


Figure 6.3: LDBA expressing the LTL formula in (4.6).

The scenario of interest in this chapter is to train a deep neural network that can autonomously control a rover to accomplish a safety-critical complex task on Mars. Given mission tasks in the form of LTL properties, we convert the LTL properties into their corresponding LDBAs so that we can feed them into the modular DDPG

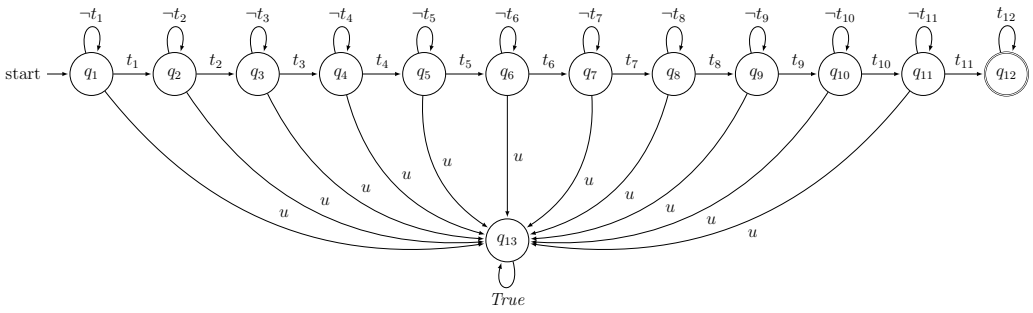


Figure 6.4: LDBA expressing the LTL formula in (6.3).

algorithm. For each actor and critic network, we used a feedforward net with 2 hidden layers and 400 ReLU units in each layer.

Presumably, from orbiting satellite data, we assume that the highest possible disturbance caused by different factors (such as sand storms) on the rover motion is known. This assumption can be set to be very conservative given the fact that there might be some unforeseen factors that were not captured by the satellite.

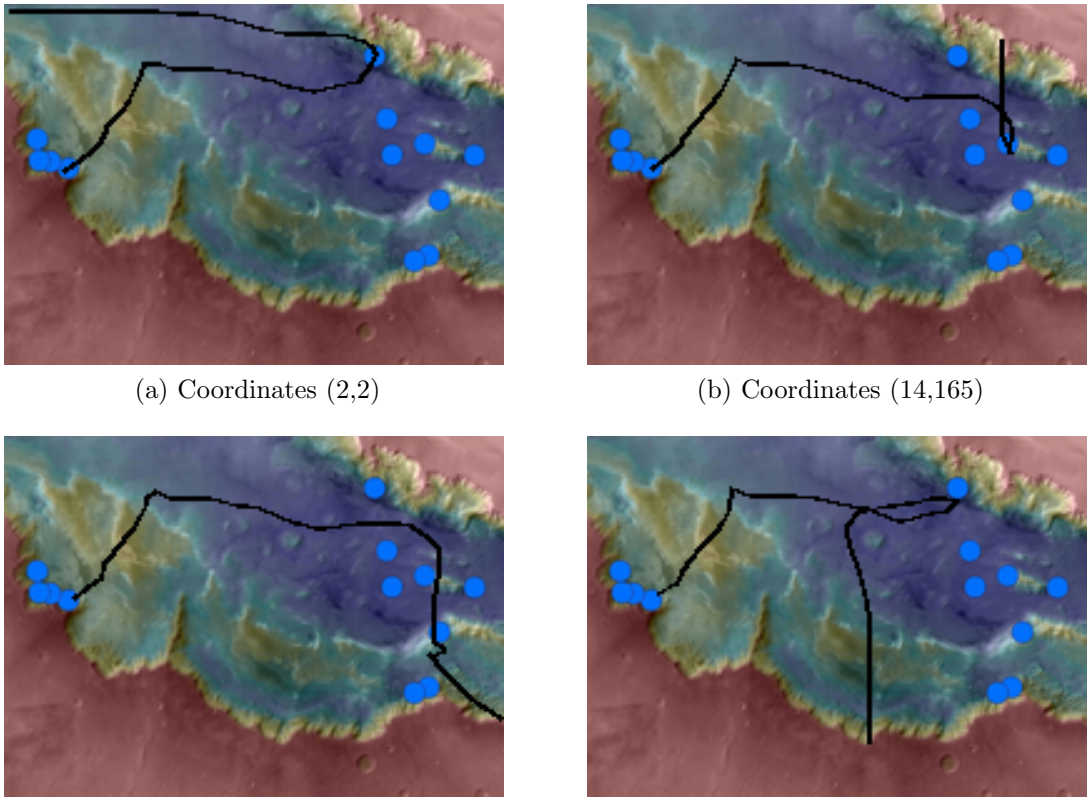
6.3.1 MDP structure

For each image, let its entire area be the MDP state space $\mathcal{S}$, where the rover location is a single state $s \in \mathcal{S}$. At each state $s \in \mathcal{S}$, the rover has a continuous range of actions $\mathcal{A} = [0, 2\pi)$: when the rover takes an action it moves to another state (e.g., s') towards the direction of the action and within a range that is randomly drawn from $(0, D]$, unless the rover hits the boundary of the image which restart the exploration.

Note that in the first experiment (Fig. 6.2. a), when the rover is deployed to its real mission, the precise landing location is not known. Therefore, we should encompass some randomness in the initial state s_0. However, in the second experiment (Fig. 6.2. b) the rover has already landed and it starts its mission from a known and fixed point.

The dimensions of the area of interest in Fig. 6.2. a are 456.98×322.58 km and

in Fig. 6.2. b are 746.98×530.12 m. Other parameters in this numerical example have been set as $D = 2$ km for Melas Chasma, $D = 10$ m for the Victoria crater. We used the satellite image itself as the black-box MDP for the algorithm. Note that the rover only needs its current state (coordinates in this case), the state label (without seeing the entire map) and the LTL task (Algorithm 6.1, line 11). The given maps in this chapter are for illustration purposes and for elucidating the performance of the output policy.



(a) Coordinates (2,2)

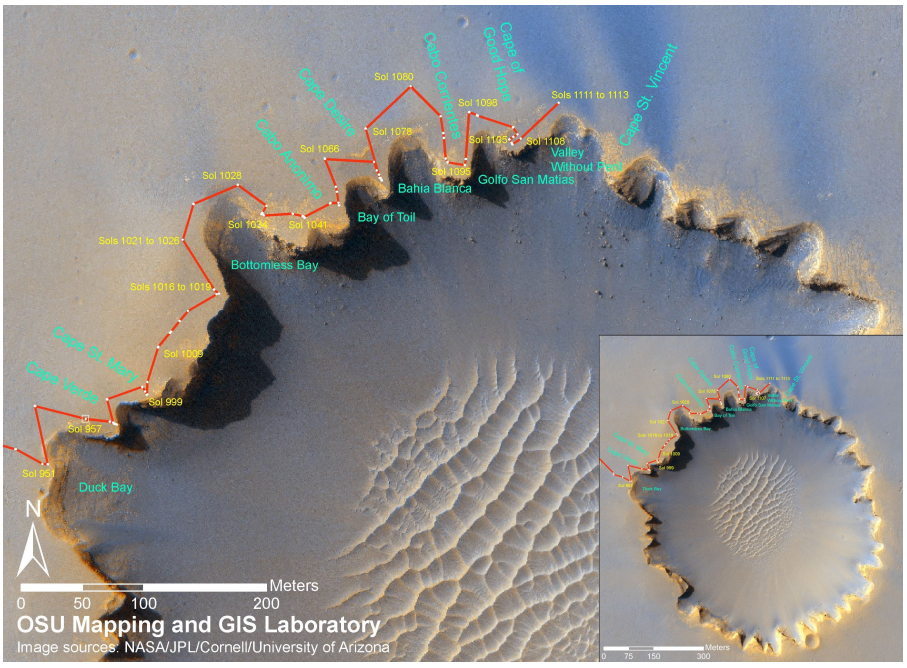
(b) Coordinates (14,165)

(c) Coordinates (113,199)

(d) Coordinates (122,113)

Figure 6.5: Generated paths by modular DDPG in Melas Chasma [8].

(a)



(b)

Figure 6.6: (a) Generated path by modular DDPG around the Victoria crater [8] vs (b) the actual Opportunity rover traverse map.

6.3.2 Specifications

The first control objective over Melas Chasma (Fig. 6.2. a) is expressed by the following LTL formula:

$$\Diamond(t_1 \wedge \Diamond t_2) \wedge \Box(t_2 \rightarrow \Box t_2) \wedge \Box(u \rightarrow \Box u), \tag{6.2}$$

where t_1 stands for "target 1", t_2 stands for "target 2" and u stands for "unsafe" (the red region in Fig. 6.2. d). Target 1 corresponds to the RSL (blue dots) on the right with a lower risk of the rover going to unsafe region, whereas the "target 2" label goes on the left RSL that are a bit riskier to explore. Conforming to (6.2), the rover has to visit any of the right dots at least once and then proceed to the left dots, while avoiding unsafe areas. From (6.2) we build the associated Büchi automaton as in Fig. 6.3.

The mission task for the Victoria crater is expressed by the following LTL formula:

$$\begin{aligned} &\Diamond(t_1 \wedge \Diamond(t_2 \wedge \Diamond(t_3 \wedge \Diamond(t_4 \wedge \Diamond(... \wedge \Diamond(t_{12}))))) \wedge \\ &\Box(t_{12} \rightarrow \Box t_{12}) \wedge \Box(u \rightarrow \Box u), \end{aligned} \tag{6.3}$$

where t_i represent the "i-th target", and u represents "unsafe". The i-th target in Fig. 6.2. c is the i-th red circle from the bottom left along the crater rim. According to (6.3) the rover is required to visit the checkpoints from the bottom left to the top right sequentially, while not falling into the crater, mimicking the actual path in Fig. 6.2. b. From (6.3), we can build the associated Büchi automaton as shown in Fig. 6.4.

Table 6.1: Modular DDPG Success Rates [8].

Case Study	Algorithm	Success Rate	Fail Rate
Melas Chasma	Standard DDPG	21.4%	78.6%
	Modular DDPG	98.8%	1.2%
Victoria Crater	Standard DDPG	0%	100%
	Modular DDPG	100%	0%

6.3.3 Simulation Results

All simulations have been carried out on a machine with an Intel Xeon 3.5GHz processor and 16GB of RAM, running Ubuntu 18. In the first experiment, we have employed 4 actor-critic neural network sets and ran simulations for 10,000 episodes. We have then tested the trained network for all safe starting positions across 200 runs. Our algorithm has achieved a success rate of 98.8% across 18,202 landing positions. Fig. 6.5 gives the path generated by our algorithm.

In the second experiment, we have used 13 actor-critic neural network sets. We have run simulations for a total of 17,000 episodes, at which point it had already converged. We have then tested the trained network across 200 runs. Our algorithm has achieved a success rate of 100% across all runs starting from t_1. Fig. 6.6 shows a generated path that is mostly curved away from the crater due to the presence of a negative reward, as described before.

We have employed stand-alone DDPG as a baseline, however, the performance has been too poor and the agents have been unable to navigate efficiently regardless of the number of episodes. In the first experiment, the rate of trials ending in unsafe (fail) state q_4 and the rate of trials in which the agent finds a path satisfying the LTL specification are given in Table 6.1. Statistics are taken over 200 different starting positions after training 10,000 episodes. In the second experiment, in order to replicate the mission of the Mars rover Opportunity, we have fixed the starting position. As mentioned before, after training, our algorithm has been able

to satisfy the LTL property with a success rate of 100% across 200 trials. After the same number of training episodes, standard DDPG has failed to synthesise a stable policy with LTL-satisfying traces (Table 6.1). In terms of potential benchmarking, we are unaware of any literature that can provide a one-shot learning baseline for such complex sequential tasks.

6.4 Implementation Details

We start our training for each automaton state only once our buffer size becomes large enough (e.g. buffer size of 4096), as a small buffer size would introduce more correlation in the data. This seems to slow down the training in the initial stage, but results in faster learning of the policies after the initial stage.

Also, we initially apply our algorithm as described with the state dimension being the x and y coordinates along with the automaton states, action dimension being the angle normalized to the range of $[-1, 1]$, where -1 represent 0 degree and 1 represents 2π. However, we find that the learning algorithm tends to perform very poorly and there exists a tendency of predicting actions at the boundary of -1 and 1. We believe that this incident occurs because the measure of the angle is circular, where 0 degree is essentially the same as 2π degree, and the learning algorithm is unaware of such property, and hence, it will be trapped in the local minimum. To resolve this issue, instead of predicting the angle for the action dimension, we instead predict the *sin* and *cos* of the angle. This approach works better as the *sin* and *cos* value of 0 and 2π is the same, and hence the circular property is preserved. We implement this approach by simply predicting two dimensions of range $[-1, 1]$ where their value is normalized such that the sum of their squares is equal to 1 as $sin^2(x) + cos^2(x) = 1$ for any x.

6.4.1 Instability of DDPG algorithm

Training the DDPG algorithm is quite challenging, and in our case, we find that the DDPG algorithm is only able to traverse to the next automaton state successfully at approximately 60-70% of the time. Therefore, the probability of reaching the i-th state is at most $0.7^{(i-1)}$. While this is not an issue for the Melas Chasma experiment (with only four states in the automaton), this causes great instability for the Victoria crater task. A solution to this problem is to stop the training for each set of DDPG nets once they become stable. However, since the DDPG net of each state of the automaton is not independent, once a DDPG net stops training, it is not able to get new updates from the DDPG net that it depends on, i.e. the next DDPG net in the automaton state.

[194] showed that applying Stochastic Weight Averaging (SWA) [195] to DDPG can improve its stability. SWA is a technique that allows for solutions to be found with better generalisation in supervised and semi-supervised learning. SWA is based on averaging the weights collected during training with an SGD-like method. In supervised learning, the weights are collected at the end of each training epoch. [195] uses a constant or cyclical learning rate schedule to prevent the optimization to converge to a single solution and continue to explore the region of high performing networks.

In order to apply SWA to DDPG algorithms, [194] introduces the frequency of updating the SWA weights. In our work, to initialize the weights we use the weights of the model that was trained until it is able to reach the next automaton state 8 times in a row. Then we apply SWA for the weights of both actor and critic networks.

6.4.2 Catastrophic forgetting

Catastrophic forgetting is the act of overwriting previous knowledge about a task when a new task is learnt. While our agent manages to become stable after applying the SWA algorithm, it started to lose accuracy after 20,000 episodes. We believe that this is due to the algorithm forgetting how to avoid unsafe regions as the experience set is filled with only successful runs. To resolve this issue, we increase the initial samples for the replay buffer to 16384 (i.e. 2^{14}) samples before we start the training. In addition to that, we separate the experience set into successful and unsuccessful experience set for each automaton state. We then sample from both replay buffers at each epoch at a fixed ratio to be tuned. We found that by separating the replay buffer and increasing the initial replay buffer sample, the algorithm is able to maintain stability after 20,000 episodes.

6.5 Summary

We discussed the first deep RL scheme for hierarchical continuous-state continuous-action decision-making problems with temporal constraints. These problems are composed of interrelated sub-problems, that in turn might have their own sub-problems. Although the optimal decision making for each sub-problem can be effortlessly done, the original problem is quite hard to be tackled holistically, even with state-of-the-art techniques. We have employed LTL to specify these interrelations and to assist the agent to find an optimal policy in a one-shot learning scheme.

Chapter 7

Cautious Reinforcement Learning

This chapter presents the concept of an adaptive "safe padding" that forces RL to synthesise optimal control policies while ensuring safety during the learning process. In a number of RL applications the safety of the agent is particularly important, e.g. expensive robotic platforms or robots that work in proximity of humans. Hence, researchers are paying increasing attention not only to maximise the long-term task-driven reward, but also to damage avoidance. 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. Later in this chapter experimental results are provided to showcase the performance of the proposed method.

7.1 Introduction

As seen in the previous chapters, RL is an algorithm with no supervision that can be used to train an agent to interact with an unknown environment, and the dynamics of the interaction are often assumed to be an MDP. 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 games 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 in general is an active area of research focusing on the training of agents with guarantees on safety [70]. 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" [10, 12, 61–66, 196]. Recent approaches on this problem [155–160] are either computationally expensive or require explicit, strong assumptions about the model of agent-environment interactions.

In this chapter, we take a step back from currently used exploration methods and recall that RL is originally inspired by cognitive and behavioural psychology [197]. 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 manoeuvrers, namely exploring new sequences of state-action pairs. It is interesting to notice that the manoeuvrers 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 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 [70]. In this context, the difficulty is mapping complex, possibly long-term, sequential tasks to an appropriate reward structure [71]. If done incorrectly, not only the agent may visit unsafe states during learning, but also the outcome of learning might be

unexpectedly sub-optimal. As an extension of ongoing LCRL research [2, 12], in this work, we employ LTL [83] as a formal reward shaping technique to specify task-related goals [82]. Recall that in LCRL, we convert a given LTL formula into a GBA-based LDBA that expresses the property [93], then translate the automaton into a state-adaptive reward structure. The obtained reward structure results in policies that maximised the probability of verifying the given LTL formula. The LCRL architecture 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 chapter will be separately accounted for during training by means of safe padding. Technically, we propose 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.

In summary, in this chapter, 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.

7.2 Cautious RL

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 7.1 (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) $P_a : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, that might not be accurate in the 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 3.1: 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 [198, 199] 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 7.1, 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 expand 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 7.1). 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

$$O_{safe}(s) = \{x \in O(s),\ q \xrightarrow{L(x)} q' \notin \Delta_{\mathbb{N}}\}, \tag{7.1}$$

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 [31]:

$$\begin{aligned} u_k(x) &= \min_{a \in \mathcal{A}} \sum_{x'} P_a(x, a, x') u_{k+1}(x'),\ k = H-1, ..., 0 \\ u_H(x) &= \mathbb{1}_{O_{safe}}(x), \end{aligned} \tag{7.2}$$

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. $u_0 = \mathbb{P}_{\min}(\square^{\leq H}\ O_{safe})$. 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:

$$U_H(s^{\otimes}, a) = 1 - \sum_{s'} P_a(s, a, s') u_0(s'), \tag{7.3}$$

where $s^{\otimes} = (s, q)$. As assumed in Problem 7.1, the agent dynamics P_a is then updated considering the Maximum Likelihood Estimation (MLE) for the mean $P_a(s, a, s')$ [198, 199] as:

$$P_a(s, a, s') \leftarrow \frac{\psi(s, a, s')}{\Psi(s, a)}. \tag{7.4}$$

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 (7.3) only depends on state s and action a, and not the automaton state q.

Remark 7.1. *Note that for each state s, the Bellman updates in* (7.2) *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* (7.2) *is limited.*

7.2.1 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, also known as LCRL. 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) [33] to maximize the expected cumulative return as in (2.3). 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 $s^{\otimes}$:

$$Q(s^{\otimes}, a) \leftarrow Q(s^{\otimes}, a) + \mu[R(s^{\otimes}, a) + \gamma \max_{a' \in \mathcal{A}^{\otimes}} (Q(s^{\otimes'}, a')) - Q(s^{\otimes}, a)], \tag{7.5}$$

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 $s^{\otimes}$ as in (3.1), $0 \leq \gamma \leq 1$ is the discount factor, and $s^{\otimes'}$ is the state

Algorithm 7.1: 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 $Q : \mathcal{S}^{\otimes} \times \mathcal{A}^{\otimes} \to \mathbb{R}^{+}$
initialize *episode-number* := 0
initialize *iteration-number* := 0
while Q *is not converged* **do**
    episode-number ← *episode-number* + 1
    $s^{\otimes} = (s_0, q_0)$
    while $(q \notin \mathcal{Q}_{sink} :\ s^{\otimes} = (s, q))\ \wedge\ (\textit{iteration-number} < \textit{it\_threshold})$ **do**
        iteration-number ← *iteration-number* + 1
        # **pessimistic learner**
            calculate $U_H(s^{\otimes}, a)$ using P_a as in (7.3)
            generate $\mathcal{A}_p^H(s^{\otimes})$ as in (7.6)
            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 (7.4)
            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

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^* [33]. 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 [2, 12] 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 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

$$\mathcal{A}_p^H(s^{\otimes}) = \{a \in \mathcal{A}^{\otimes} : U_H(s^{\otimes}, a) < p_{critical}\}. \tag{7.6}$$

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

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

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

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

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

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 [33], 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:

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

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 [33]. 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. 7.2).

7.3 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 to 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 $\mathcal{A} = \{\mathit{left}, \mathit{right}, \mathit{up}, \mathit{down}, \mathit{stay}\}$ 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.

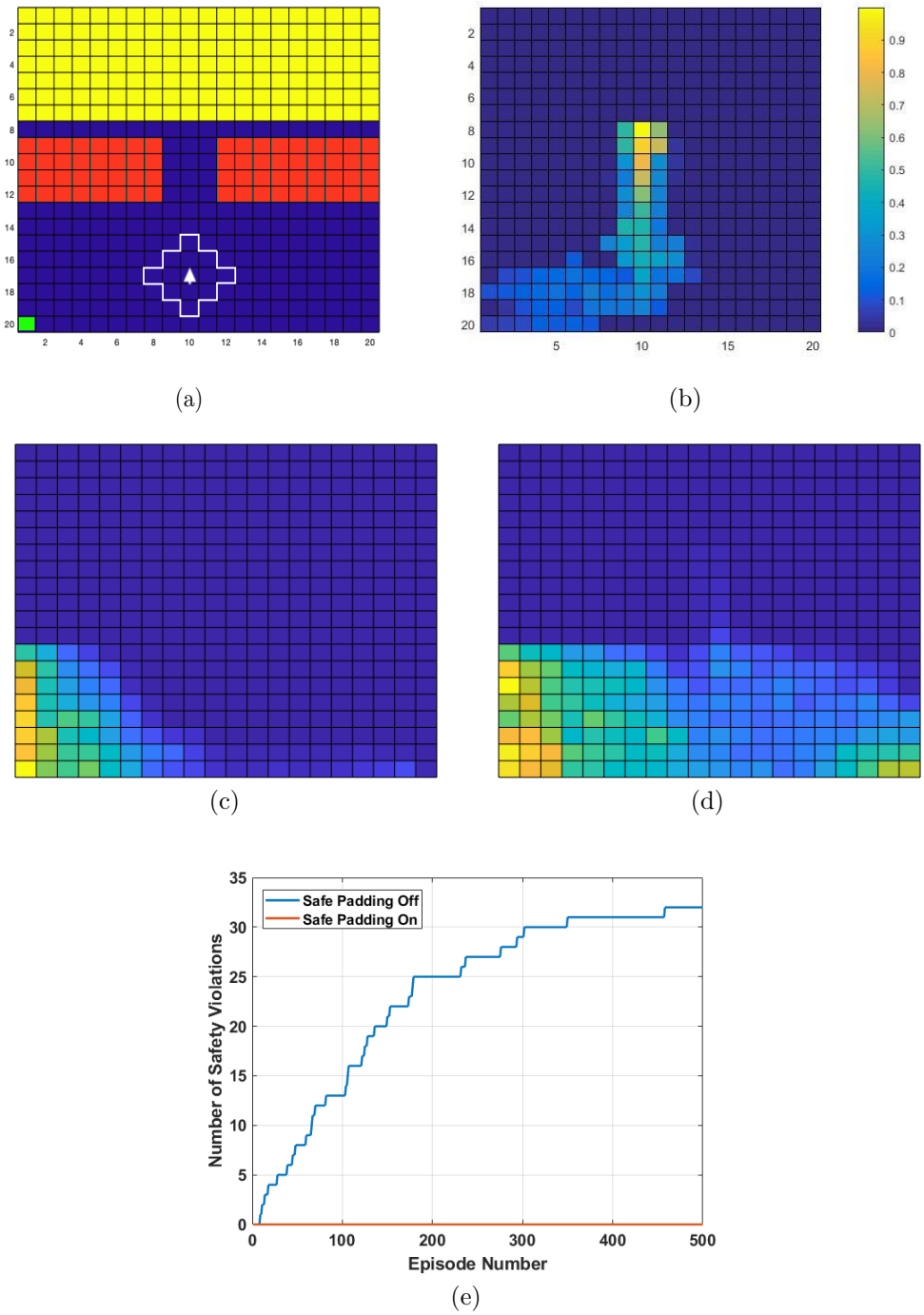


Figure 7.1: (a) slippery grid world and agent, represented by an arrow surrounded by an observation area. Labelling is yellow: *target*, red: *unsafe*, blue: *safe*, and green is the initial state s_0; (b) final value function $V(s)$, (c)–(d) state visitation number $v(s) = \sum_a \Psi(s, a)$ vs. time 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.

To get to the target state the agent has to cross a bridge (Fig. 7.1a) 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.

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:

$$\Diamond target \wedge \Box \neg unsafe. \tag{7.7}$$

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. 7.1c–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. 7.1b) according to which the safe policy is later generated.

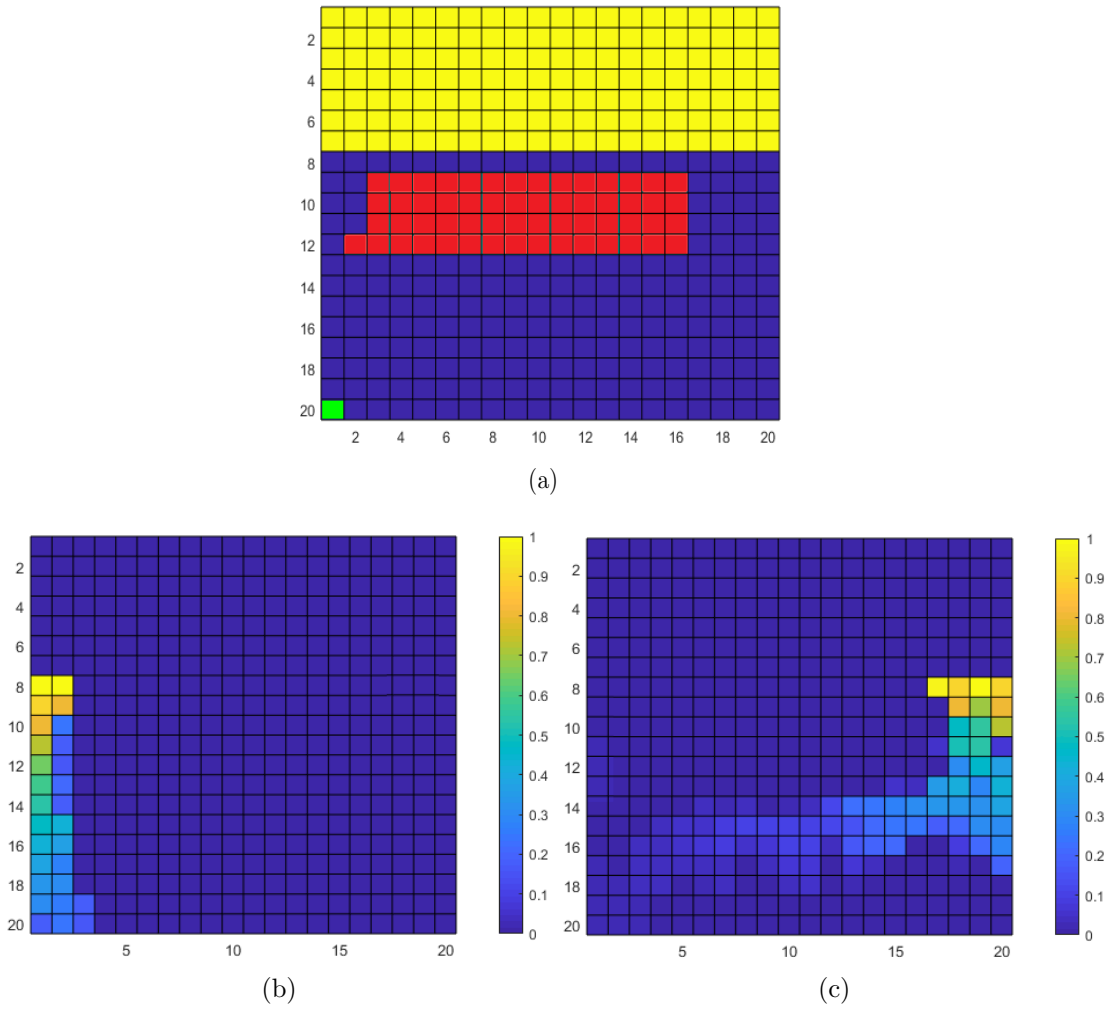
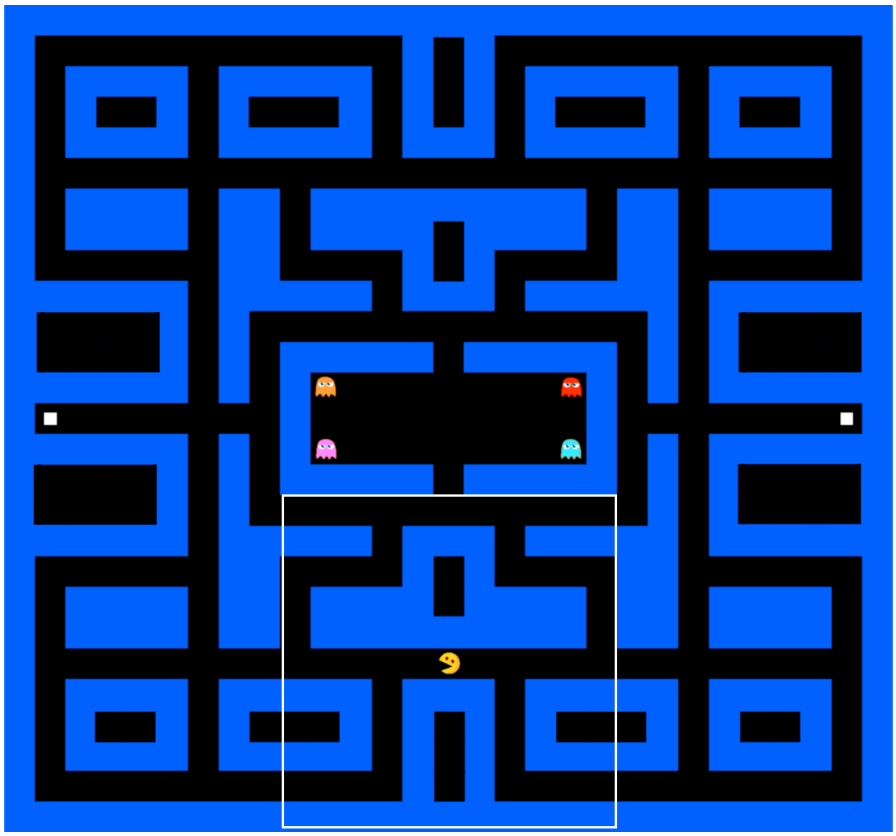


Figure 7.2: Safety and performance trade-off: (a) slippery grid world with two options to satisfy formula (7.7), where labelling is yellow: *target*, red: *unsafe*, blue: *safe*, and green is the initial state s_0; (b) value function $V(s)$ without safe padding; and (c) value function with safe padding (cautious RL).

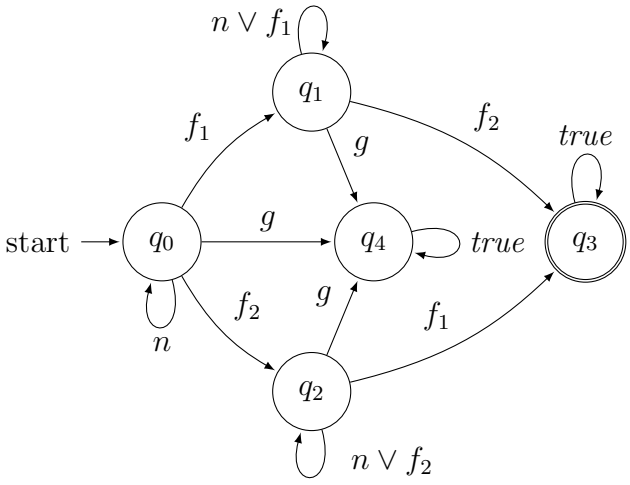
Furthermore, note that with Cautious RL the agent is focused on those parts of the state space that are most relevant to the satisfaction of the given LTL property.

There was no single incident of going to unsafe in this experiment even with such a limited observation radius (Table 7.1). 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

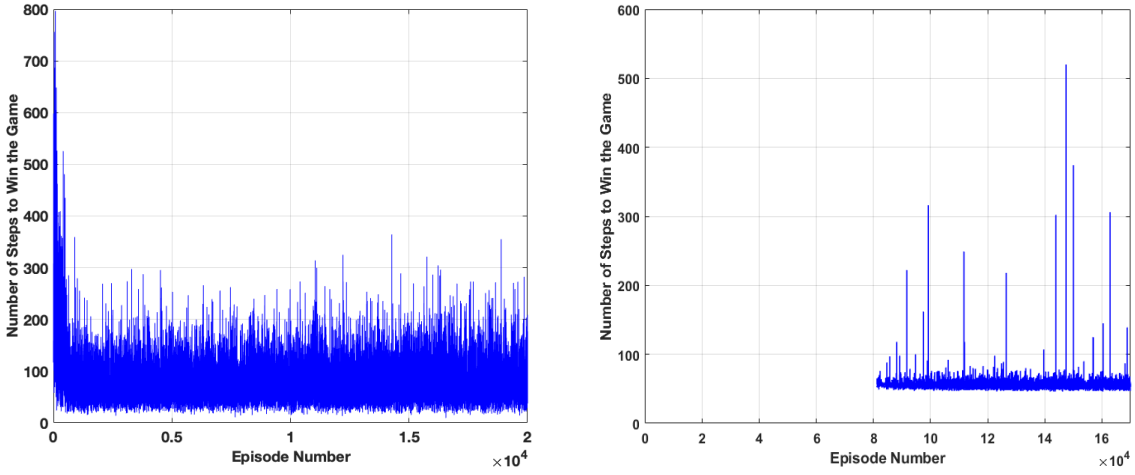


(a)



(b)

Figure 7.3: (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 (7.8).



(a) (b)

Figure 7.4: (a) Number of steps to complete the game with safe padding on (cautious RL), (b) and with safe padding off.

Table 7.1: The proportion of the number of times that the agent ended in unsafe (fail) states, and the proportion of the 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%

(Fig. 7.3a). In order to win the agent has to collect all tokens without being caught by ghosts:

$$\Diamond[(f_1 \wedge \Diamond f_2) \vee (f_2 \wedge \Diamond f_1)] \wedge \Box\neg g, \tag{7.8}$$

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. 7.3b. 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. 7.4a gives the results of learning with safe padding on and Fig. 7.4b off. Note that with the safe padding on, the agent was able to successfully escape the ghosts even from the beginning, with the cost of a 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 7.1).

7.4 Summary

We have proposed Cautious RL, 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.

Chapter 8

Conclusions and Future Work

This chapter summarises the dissertation by discussing key contributions and also the research directions that are currently under investigation.

8.1 Conclusions

In this work, we have proposed a general learning method to guide an RL agent, by expanding the agent domain knowledge about the environment by means of an LTL property. This additional knowledge, as we have observed in experiments, boosts the agent learning of the globally optimal policy. Further, we have shown that in finite MDPs we can efficiently determine the probability that is associated with the satisfaction of the LTL property that enables us to quantify the safety of the generated optimal policy at any given state.

We have shown that converting the LTL property to a GBA-based LDBA results in a significantly smaller automaton than DRA alternatives, which increases the convergence rate of RL. 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 due to the simpler accepting conditions of LDBA. Much like the way we synchronised the states of LDBA with the states of the MDP, we have shown that synchronising a Kripke structure with the LDBA allows us to handle time-varying periodic environments on-the-fly. This particularly becomes important when we employed this synchronised LDBA-Kripke automaton as an infrastructure for the agent to transfer its learning over the dimension of time and to overcome the curse of dimensionality.

For uncountably infinite-state MDPs we introduced LCNFQ, the first RL algorithm that can offer LTL policy synthesis in a continuous-state MDP. LCNFQ is model-free, meaning that the learning only depends on the sample experiences that the agent gathered by interacting and exploring the MDP. Further, the sample set can be small thanks to the generalisation that neural nets offer. The core engine in LCNFQ is very flexible and can be extended to the most recent developments in RL literature.

Following further upon LCNFQ we proposed a fully-unsupervised one-shot online learning architecture for deep RL, where the learner is presented with an LTL formula that acts as a composable high-level mission task in a continuous-state and continuous-action MDP. Recall that the LTL property is a formal, un-grounded, and symbolic representation of the task and of its components. Without requiring any supervision, each component of the LTL property systematically structures any complex mission task into low-level, achievable task "modules". The LTL property essentially acts as a high-level unsupervised guide for the agent, whereas the low-level planning is handled by a deep actor-critic architecture. The proposed algorithm is the first deep RL scheme for hierarchical continuous-state continuous-action decision-making problems with temporal constraints. These problems are composed of interrelated sub-problems, that in turn might have their own sub-problems. The

natural method is to break down the original problem into sub-problem that can be effortlessly solved. Though, the original problem is quite hard to be tackled holistically, even with state-of-the-art techniques. As shown in the experiments, by employing LTL to specify these interrelations we are able to assist the agent to find an optimal policy in a one-shot learning scheme, and subsequently, synthesise a closely-optimal policy.

In summary, we showed

- Model-free RL with discounted rewards can be employed provably-correct for LTL policy synthesis and quantitative model-checking in finite MDPs.

  Impact:

  - The use of model-free RL, significantly widened the applicability of RL-based LTL synthesis approaches, more specifically to problems with large state and action space cardinality.
  - Model-free RL makes quantitative model-checking easier to implement and faster to converge than model-based methods.

- LTL policy synthesis can be achieved using model-free (deep) RL in continuous-state continuous-action MDPs with efficient training time, sample complexity, and success rate.

  Impact:

  - LTL synthesis can now be done directly in continuous-state MDPs, without the need to discretise states or building a model.
  - In continuous-state continuous-action MDPs, when dealing with LTL synthesis problems, dynamics accuracy is not lost thanks to the fact that action space is not discretised, as opposed to conventional discretisation and abstraction methods.

- The use of model-free (deep) RL, improved sample complexity and training time of LTL synthesis in continuous MDPs.

- Owing to its succinct structure, the use of LDBA in LTL synthesis increased (deep) RL rate of convergence and its scalability comparing to methods that use DRA, both in finite and infinite MDPs.

8.2 Future Work

There are a number of possible directions towards which the work in this dissertation can be developed further.

8.2.1 Abstraction Techniques for Large-Scale MDPs

LCRL synthesises control policies by focusing on those parts of the state space that are relevant to the satisfaction of a given LTL property. This is proved empirically to be much faster than other conventional methods such as DP [200] or other approximate variants of it, e.g. [105]. This improvement in speed can be pushed forward by abstracting the MDP into a simpler MDP with smaller state space, making the exploration easier for LCRL. Interesting papers to start with are [109, 154, 201].

8.2.2 LCRL for Partially Observable MDPs

Partially Observable Markov Decision Processes (POMDPs) provide a modelling framework for a variety of sequential decision making under uncertainty [202]. The states are not directly observable in a POMDP, and decision making has to be performed based on a belief upon certain observations. The belief is the agent inference of the likelihood of being in a certain state. Although, solving a POMDP exactly, i.e. synthesizing a policy ensuring optimality is undecidable [203],

there exist several approximation methods, e.g. [204]. We would like to develop a model-free RL algorithm that can synthesise policies with respect to temporal logic properties over POMDPs.

8.2.3 Multi-Objective LCRL

Multi-objective RL problems differ from conventional RL setups in having two or more (possibly conflicting) objectives, each with its own reward signal. The goal in large-scale multi-objective RL problems is to find an approximation of the Pareto front. We are interested to explore whether we can have one (or more) reward signal(s) apart from the reward that we consider for the LTL property in LCRL, and then convert the problem into finding the Pareto front for these objectives. There is a large body of research in this area both in RL community and formal methods community, e.g. [205–210].

8.2.4 Multi-agent LCRL

Imagine that you have a heterogeneous set of robots, each is able to execute a very specific task. For any given LTL property, multi-agent LCRL looks for a way to distribute different sub-tasks to different agents. It is not a trivial problem as a number of factors such as sub-task sequentiality, agents availability, etc. are important. Interesting references in this direction are [211–215].

8.2.5 RL in CEGIS

CEGIS (Counter-example Guided Inductive Synthesis) [216] is an architecture that employs SAT-solving and can be used for controller synthesis. However, CEGIS only looks for satisfaction, disregarding optimality. We believe that the employment of RL in the internal engine of CEGIS can (1) introduce the notion of optimality to

the output of CEGIS, and (2) possibly can increase the convergence rate of CEGIS in some cases. Further to this, LCRL leverages temporal logic in learning, which is very much aligned with CEGIS architecture that hinges on such specifications.

Bibliography

[1] M. Hasanbeig, A. Abate, and D. Kroening, "Logically-constrained neural fitted Q-iteration," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 2012–2014.

[2] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee, "Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees," in *Proceedings of the 58th Conference on Decision and Control*. IEEE, 2019, pp. 5338–5343.

[3] M. Hasanbeig, A. Abate, and D. Kroening, "Cautious reinforcement learning with logical constraints," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

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

[5] M. Hasanbeig, N. Yogananda Jeppu, A. Abate, T. Melham, and D. Kroening, "DeepSynth: Automata synthesis for automatic task segmentation in

deep reinforcement learning," in *AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence, 2021.

[6] M. Giacobbe, M. Hasanbeig, D. Kroening, and H. Wijk, "Shielding Atari games with bounded prescience," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2021.

[7] M. Cai, M. Hasanbeig, S. Xiao, A. Abate, and Z. Kan, "Modular deep reinforcement learning for continuous motion planning with temporal logic," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2021.

[8] L. Z. Yuan, M. Hasanbeig, A. Abate, and D. Kroening, "Modular Deep Reinforcement Learning with Temporal Logic Specifications," *arXiv preprint arXiv:1909.11591*, 2019.

[9] M. Hasanbeig, N. Yogananda Jeppu, A. Abate, T. Melham, and D. Kroening, "DeepSynth: Automata synthesis for automatic task segmentation in deep reinforcement learning," in *10th Workshop on Synthesis*. International Conference on Computer-Aided Verification, 2021.

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

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

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

[13] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia, "A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications," in *IEEE Conference on Decision and Control (CDC)*. IEEE, 2014, pp. 1091–1096.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press Cambridge, 1998, vol. 1.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," *Advances in Neural Information Processing Systems*, vol. 19, p. 1, 2007.

[17] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[18] M. Hasanbeig, "Multi-agent learning in coverage control games," Master's thesis, University of Toronto (Canada), 2016.

[19] Z. Zhou, X. Li, and R. N. Zare, "Optimizing chemical reactions with deep reinforcement learning," *ACS Central Science*, vol. 3, no. 12, pp. 1337–1344, 2017.

[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[21] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *ACM Workshop on Networks*. ACM, 2016, pp. 50–56.

[22] Y. Kantaros, T. Carpenter, S. Park, R. Ivanov, S. Jang, I. Lee, and J. Weimer, "VisionGuard: Runtime detection of adversarial inputs to perception systems," *arXiv preprint arXiv:2002.09792*, 2020.

[23] S. Rahili and W. Ren, "Game theory control solution for sensor coverage problem in unknown environment," in *IEEE Conference on Decision and Control (CDC)*. IEEE, 2014, pp. 1173–1178.

[24] M. Hasanbeig and L. Pavel, "Distributed coverage control by robot networks in unknown environments using a modified EM algorithm," *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 11, no. 7, pp. 805–813, 2017.

[25] F. Memarian, Z. Xu, B. Wu, M. Wen, and U. Topcu, "Active task-inference-guided deep inverse reinforcement learning," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 1932–1938.

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

[27] M. Hasanbeig and L. Pavel, "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.

[28] ——, "From game-theoretic multi-agent log linear learning to reinforcement learning," *arXiv preprint arXiv:1802.02277*, 2018.

[29] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[30] M. van Otterlo and M. Wiering, "Reinforcement learning and Markov decision processes," in *Reinforcement Learning*. Springer, 2012, pp. 3–42.

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

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

[33] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[34] T. Jaakkola, M. I. Jordan, and S. P. Singh, "Convergence of stochastic iterative dynamic programming algorithms," in *Advances in Neural Information Processing Systems*, 1994, pp. 703–710.

[35] D. Bertsekas, "Convergence of discretization procedures in dynamic programming," *IEEE Transactions on Automatic Control*, vol. 20, no. 3, pp. 415–419, 1975.

[36] M. Prandini and J. Hu, "A stochastic approximation method for reachability computations," in *Stochastic Hybrid Systems*. Springer, 2006, pp. 107–139.

[37] F. Dufour and T. Prieto-Rumeau, "Approximation of Markov decision processes with general state space," *Journal of Mathematical Analysis and Applications*, vol. 388, no. 2, pp. 1254–1267, 2012.

[38] S. E. Z. Soudjani, C. Gevaerts, and A. Abate, "FAUST^2: Formal Abstractions of Uncountable-STate STochastic Processes," in *International Conference on*

Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2015, pp. 272–286.

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

[40] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Machine Learning*, vol. 49, no. 2-3, pp. 291–323, 2002.

[41] J. Stachurski, "Continuous state dynamic programming via nonexpansive approximation," *Computational Economics*, vol. 31, no. 2, pp. 141–160, 2008.

[42] H. M. Bayer and P. W. Glimcher, "Midbrain Dopamine neurons encode a quantitative reward prediction error signal," *Neuron*, vol. 47, no. 1, pp. 129–141, 2005.

[43] W. Schultz, "Predictive reward signal of Dopamine neurons," *Journal of Neurophysiology*, vol. 80, no. 1, pp. 1–27, 1998.

[44] K. D'Ardenne, S. M. McClure, L. E. Nystrom, and J. D. Cohen, "BOLD responses reflecting Dopaminergic signals in the human ventral tegmental area," *Science*, vol. 319, no. 5867, pp. 1264–1267, 2008.

[45] M. Grześ, "Reward shaping in episodic reinforcement learning," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 565–573.

[46] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine learning*, vol. 49, no. 2-3, pp. 209–232, 2002.

[47] A. L. Strehl and M. L. Littman, "An analysis of model-based interval estimation for Markov decision processes," *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1309–1331, 2008.

[48] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219–245, 2000.

[49] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems*, 1996, pp. 1038–1044.

[50] D. Ormoneit and Ś. Sen, "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, no. 2, pp. 161–178, 2002.

[51] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, no. Apr, pp. 503–556, 2005.

[52] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC press, 2010, vol. 39.

[53] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[54] G. Tesauro, "TD-Gammon: A self-teaching Backgammon program," in *Applications of Neural Networks*. Springer, 1995, pp. 267–285.

[55] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[56] M. Riedmiller, "Neural fitted Q iteration-first experiences with a data efficient neural reinforcement learning method," in *European Conference on Machine Learning*, vol. 3720. Springer, 2005, pp. 317–328.

[57] H. Van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," in *Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. IEEE, 2007, pp. 272–279.

[58] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[59] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *2015 AAAI Fall Symposium Series*, 2015.

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

[61] S. P. Coraluppi and S. I. Marcus, "Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes," *Automatica*, vol. 35, no. 2, pp. 301–309, 1999.

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

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

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

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

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

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

[68] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

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

[70] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[71] S. Bharadwaj, S. Le Roux, G. Pérez, and U. Topcu, "Reduction techniques for model checking and learning in MDPs," in *Proceedings of the 26st International Joint Conference on Artificial Intelligence, Melbourne*, 2017, pp. 4273–4279.

[72] F. Memarian, W. Goo, R. Lioutikov, U. Topcu, and S. Niekum, "Self-supervised online reward shaping in sparse-reward environments," *arXiv preprint arXiv:2103.04529*, 2021.

[73] P. Abbeel, *Apprenticeship Learning and Reinforcement Learning With Application to Robotic Control*. Stanford University, 2008.

[74] Y. Aytar, T. Pfaff, D. Budden, T. L. Paine, Z. Wang, and N. de Freitas, "Playing hard exploration games by watching YouTube," *arXiv preprint arXiv:1805.11592*, 2018.

[75] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Springer Handbook of Robotics*. Springer, 2008, pp. 1371–1394.

[76] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[77] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold *et al.*, "Deep Q-learning from demonstrations," *arXiv preprint arXiv:1704.03732*, 2017.

[78] I.-A. Hosu and T. Rebedea, "Playing Atari games with deep reinforcement learning and human checkpoint replay," *arXiv preprint arXiv:1607.05077*, 2016.

[79] Z. Lipton, X. Li, J. Gao, L. Li, F. Ahmed, and L. Deng, "BBQ-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems," *arXiv preprint arXiv:1711.05715*, 2017.

[80] T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. van Hasselt, J. Quan, M. Večerík *et al.*, "Observe and look further: Achieving consistent performance on Atari," *arXiv preprint arXiv:1805.11593*, 2018.

[81] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," *arXiv preprint arXiv:1709.10089*, 2017.

[82] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model Checking*. MIT press, 2018.

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

[84] A. P. Nikora and G. Balcom, "Automated identification of LTL patterns in natural language requirements," in *International Symposium on Software Reliability Engineering*. IEEE, 2009, pp. 185–194.

[85] R. Yan, C.-H. Cheng, and Y. Chai, "Formal consistency checking over specifications in natural languages," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1677–1682.

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

[87] D. Precup, "Temporal abstraction in reinforcement learning," Ph.D. dissertation, University of Massachusetts Amherst, 2001.

[88] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine Learning*, vol. 49, no. 2-3, pp. 209–232, 2002.

[89] C. Daniel, G. Neumann, and J. Peters, "Hierarchical relative entropy policy search," in *Artificial Intelligence and Statistics*, 2012, pp. 273–281.

[90] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in Neural Information Processing Systems*, 2016, pp. 3675–3683.

[91] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, J. Agapiou *et al.*, "Strategic attentive writer for learning macro-actions," in *Advances in Neural Information Processing Systems*, 2016, pp. 3486–3494.

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

[93] C. Baier, J.-P. Katoen, and K. G. Larsen, *Principles of Model Checking*. MIT Press, 2008.

[94] S. Safra, "On the complexity of Omega-automata," in *Foundations of Computer Science, 1988., 29th Annual Symposium on*. IEEE, 1988, pp. 319–327.

[95] N. Piterman, "From nondeterministic Büchi and Streett automata to deterministic parity automata," in *ACM/IEEE Symposium on Logic in Computer Science*. IEEE, 2006, pp. 255–264.

[96] R. Alur and S. La Torre, "Deterministic generators and games for LTL fragments," *ACM Transactions on Computational Logic*, vol. 5, no. 1, pp. 1–25, 2004.

[97] S. Sickert, J. Esparza, S. Jaax, and J. Křetínskỳ, "Limit-deterministic Büchi automata for linear temporal logic," in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2016, pp. 312–332.

[98] S. Sickert and J. Křetínskỳ, "MoChiBA: Probabilistic LTL model checking using limit-deterministic Büchi automata," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2016, pp. 130–137.

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

[100] D. Kini and M. Viswanathan, "Optimal translation of LTL to limit deterministic automata," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2017, pp. 113–129.

[101] E. M. Wolff, U. Topcu, and R. M. Murray, "Robust control of uncertain Markov decision processes with temporal logic specifications," in *IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 3372–3379.

[102] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[103] M. Wen, R. Ehlers, and U. Topcu, "Correct-by-synthesis reinforcement learning with temporal logic constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2015, pp. 4983–4990.

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

[105] T. Brázdil, K. Chatterjee, M. Chmelík, V. Forejt, J. Křetínskỳ, M. Kwiatkowska, D. Parker, and M. Ujma, "Verification of Markov decision processes using learning algorithms," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2014, pp. 98–114.

[106] S. Junges, N. Jansen, C. Dehnert, U. Topcu, and J.-P. Katoen, "Safety-constrained reinforcement learning for MDPs," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2016, pp. 130–146.

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

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

[109] I. S. Lee and H. Y. Lau, "Adaptive state space partitioning for reinforcement learning," *Engineering applications of artificial intelligence*, vol. 17, no. 6, pp. 577–588, 2004.

[110] G. J. Gordon, "Stable function approximation in dynamic programming," in *Machine Learning*. Elsevier, 1995, pp. 261–268.

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

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

[113] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and Z. Wang, "Time for statistical model checking of real-time systems," in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2011, pp. 349–355.

[114] K. G. Larsen, "Priced timed automata and statistical model checking," in *International Conference on Integrated Formal Methods*. Springer, 2013, pp. 154–161.

[115] R. Lassaigne and S. Peyronnet, "Approximate planning and verification for large Markov decision processes," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 457–467, 2015.

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

[117] M. Cai, H. Peng, Z. Li, and Z. Kan, "Learning-based probabilistic LTL motion planning with environment and motion uncertainties," *IEEE Trans. Autom. Control*, 2020, to appear.

[118] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak, "Omega-regular objectives in model-free reinforcement learning," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2019, pp. 395–412.

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

[120] B. Lennartson and Q. Jia, "Reinforcement learning with temporal logic constraints," in *Proceedings of the International Workshop on Discrete Event Systems (WODES)*, 2020.

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

[122] M. Kazemi and S. Soudjani, "Formal policy synthesis for continuous-space systems via reinforcement learning," *arXiv preprint arXiv:2005.01319*, 2020.

[123] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Teaching multiple tasks to an RL agent using LTL," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018, pp. 452–461.

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

[125] A. Camacho, R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "LTL and beyond: Formal languages for reward function specification in reinforcement learning," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 6065–6073.

[126] A. Lavaei, F. Somenzi, S. Soudjani, A. Trivedi, and M. Zamani, "Formal controller synthesis for continuous-space MDPs via model-free reinforcement learning," in *ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2020, pp. 98–107.

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

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

[129] K. Lesser and A. Abate, "Multiobjective optimal control with safety as a priority," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 3, pp. 1015–1027, 2018.

[130] M. Svorenova, I. Cerna, and C. Belta, "Optimal control of MDPs with temporal logic constraints," in *IEEE Conference on Decision and Control (CDC)*. IEEE, 2013, pp. 3938–3943.

[131] A. David, P. G. Jensen, K. G. Larsen, M. Mikučionis, and J. H. Taankvist, "Uppaal stratego," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 206–211.

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

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

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

[135] O. Andersson, F. Heintz, and P. Doherty, "Model-based reinforcement learning in continuous environments using real-time constrained optimization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015, pp. 2497–2503.

[136] A. L. Thomaz and C. Breazeal, "Teachable robots: Understanding human teaching behavior to build more effective robot learners," *Artificial Intelligence*, vol. 172, no. 6-7, pp. 716–737, 2008.

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

[138] N. Fulton and A. Platzer, "Verifiably safe off-model reinforcement learning," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2019, pp. 413–430.

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

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

[141] N. Fulton and A. Platzer, "Safe reinforcement learning via formal methods: Toward safe control through proof and learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

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

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

[144] L. Belzner and M. Wirsing, "Synthesizing Safe Policies under Probabilistic Constraints with Reinforcement Learning and Bayesian Model Checking," *arXiv preprint arXiv:2005.03898*, 2020.

[145] K. Polymenakos, L. Laurenti, A. Patane, J.-P. Calliess, L. Cardelli, M. Kwiatkowska, A. Abate, and S. Roberts, "Safety guarantees for iterative predictions with Gaussian processes," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 3187–3193.

[146] K. Polymenakos, N. Rontsis, A. Abate, and S. Roberts, "SafePILCO: A software tool for safe and data-efficient policy synthesis," in *Quantitative Evaluation of Systems: 17th International Conference, QEST 2020, Vienna, Austria, August 31-September 3, 2020, Proceedings*, vol. 12289. Springer, 2020, p. 18.

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

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

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

[150] S. Haesaert, S. E. Zadeh Soudjani, and A. Abate, "Verification of general Markov decision processes by approximate similarity relations and policy refinement," *SIAM Journal on Control and Optimization*, vol. 55, no. 4, pp. 2333–2367, 2017.

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

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

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

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

[155] M. Wen and U. Topcu, "Constrained cross-entropy method for safe reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 7450–7460.

[156] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "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, 2019, pp. 3387–3395.

[157] X. Li and C. Belta, "Temporal logic guided safe reinforcement learning using control barrier functions," *arXiv preprint arXiv:1903.09885*, 2019.

[158] T. M. Moldovan and P. Abbeel, "Safe exploration in Markov decision processes," *arXiv preprint arXiv:1205.4810*, 2012.

[159] M. Turchetta, F. Berkenkamp, and A. Krause, "Safe exploration in finite Markov decision processes with Gaussian processes," in *Advances in Neural Information Processing Systems*, 2016, pp. 4312–4320.

[160] S. Li and O. Bastani, "Robust model predictive shielding for safe reinforcement learning with stochastic dynamics," *arXiv preprint arXiv:1910.10885*, 2019.

[161] D. P. Bertsekas and S. Shreve, *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific, 2004.

[162] R. Durrett, *Essentials of Stochastic Processes*. Springer, 1999, vol. 1.

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

[164] N. Yoshida, E. Uchibe, and K. Doya, "Reinforcement learning with state-dependent discount factor," in *IEEE International Conference on Development and Learning*. IEEE, 2013, pp. 1–6.

[165] Q. Wei and X. Guo, "Markov decision processes with state-dependent discount factors and unbounded rewards/costs," *Operations Research Letters*, vol. 39, no. 5, pp. 369–374, 2011.

[166] R. G. Newell and W. A. Pizer, "Discounting the distant future: How much do uncertain rates increase valuations?" *Journal of Environmental Economics and Management*, vol. 46, no. 1, pp. 52–71, 2003.

[167] R. Cavazos-Cadena, E. A. Feinberg, and R. Montes-De-Oca, "A note on the existence of optimal policies in total reward dynamic programs with compact action sets," *Mathematics of Operations Research*, vol. 25, no. 4, pp. 657–666, 2000.

[168] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[169] A. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées," *Comp. Rend. Sci. Paris*, vol. 25, no. 1847, pp. 536–538, 1847.

[170] M. Riedmiller, "Concepts and facilities of a neural reinforcement learning control architecture for technical process control," *Neural Computing & Applications*, vol. 8, no. 4, pp. 323–338, 1999.

[171] L.-H. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3/4, pp. 69–97, 1992.

[172] J. Peters and J. A. Bagnell, "Policy gradient methods," *Scholarpedia*, vol. 5, no. 11, p. 3698, 2010.

[173] D. Silver, G. Lever, N. Heess, D. W. Thomas Degris, and M. Riedmiller, "Deterministic policy gradient algorithms," *International Conference on Machine Learning*, 2014.

[174] A. P. Sistla and E. M. Clarke, "The complexity of propositional linear temporal logics," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 733–749, 1985.

[175] J. Cohen, D. Perrin, and J.-E. Pin, "On the expressive power of temporal logic," *Journal of computer and System Sciences*, vol. 46, no. 3, pp. 271–294, 1993.

[176] A. Muscholl and I. Walukiewicz, "An NP-complete fragment of LTL," in *International Conference on Developments in Language Theory*. Springer, 2004, pp. 334–344.

[177] R. I. Brafman, G. De Giacomo, and F. Patrizi, "LTLf/LDLf non-Markovian rewards," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[178] G. De Giacomo, L. Iocchi, M. Favorito, and F. Patrizi, "Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications," in *International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 128–136.

[179] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer-Aided Verification (CAV)*. Springer, 2011, pp. 585–591.

[180] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "The probabilistic model checker Storm," *arXiv preprint arXiv:1610.08713*, 2016.

[181] S. Junges, N. Jansen, R. Wimmer, T. Quatmann, L. Winterer, J. Katoen, and B. Becker, "Finite-state controllers of POMDPs using parameter synthesis," *UAI*, 2018.

[182] L. Duan, D. Xu, and I. Tsang, "Learning with augmented features for heterogeneous domain adaptation," *arXiv preprint arXiv:1206.4660*, 2012.

[183] M. Harel and S. Mannor, "Learning from multiple outlooks," *arXiv preprint arXiv:1005.0027*, 2010.

[184] B. Kulis, K. Saenko, and T. Darrell, "What you saw is not what you get: Domain adaptation using asymmetric kernel transforms," in *Conference on Computer Vision and Pattern Recognition*. IEEE, 2011, pp. 1785–1792.

[185] A. Fern, S. Yoon, and R. Givan, "Approximate policy iteration with a policy language bias," in *Advances in Neural Information Processing Systems*, 2004, pp. 847–854.

[186] O. Ilghami, H. Munoz-Avila, D. S. Nau, and D. W. Aha, "Learning approximate preconditions for methods in hierarchical plans," in *International Conference on Machine Learning*. ACM, 2005, pp. 337–344.

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

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

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

[190] O. Hernández-Lerma and J. B. Lasserre, *Further Topics on Discrete-Time Markov Control Processes*. Springer Science & Business Media, 2012, vol. 42.

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

[192] A. S. McEwen, C. M. Dundas, S. S. Mattson, A. D. Toigo, L. Ojha, J. J. Wray, M. Chojnacki, S. Byrne, S. L. Murchie, and N. Thomas, "Recurring slope lineae in equatorial regions of Mars," *Nature Geoscience*, vol. 7, no. 1, p. 53, 2014.

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

[194] E. Nikishin, P. Izmailov, B. Athiwaratkun, D. Podoprikhin, T. Garipov, P. Shvechikov, D. Vetrov, and A. G. Wilson, "Improving stability in deep

reinforcement learning with weight averaging," in *UAI Workshop on Uncertainty in Deep Learning*, 2018, p. 5.

[195] B. Athiwaratkun, M. Finzi, P. Izmailov, and A. G. Wilson, "Improving consistency-based semi-supervised learning with weight averaging," *arXiv preprint arXiv:1806.05594*, vol. 2, 2018.

[196] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A Lyapunov-based approach to safe reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 8092–8101.

[197] E. L. Thorndike, *The Fundamentals of Learning*. Teachers College Bureau of Publications, 1932.

[198] A. L. Strehl, L. Li, and M. L. Littman, "Reinforcement learning in finite MDPs: PAC analysis," *Journal of Machine Learning Research*, vol. 10, no. Nov, pp. 2413–2444, 2009.

[199] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[200] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena scientific Belmont, MA, 1995, vol. 1, no. 2.

[201] L. Li, T. J. Walsh, and M. L. Littman, "Towards a unified theory of state abstraction for mdps," in *International Symposium on Artificial Intelligence and Mathematics*, 2006.

[202] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Malaysia; Pearson Education Limited,, 2016.

[203] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems," in *AAAI/IAAI*, 1999, pp. 541–548.

[204] M. Hauskrecht, "Value-function approximations for partially observable Markov decision processes," *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, 2000.

[205] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker, "Empirical evaluation methods for multiobjective reinforcement learning algorithms," *Machine learning*, vol. 84, no. 1-2, pp. 51–80, 2011.

[206] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 41–47.

[207] Y. Mukai, Y. Kuroe, and H. Iima, "Multi-objective reinforcement learning method for acquiring all Pareto optimal policies simultaneously," in *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1917–1923.

[208] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *Journal of Artificial Intelligence Research*, vol. 48, pp. 67–113, 2013.

[209] E. M. Hahn, V. Hashemi, H. Hermanns, M. Lahijanian, and A. Turrini, "Multi-objective robust strategy synthesis for interval Markov decision processes," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2017, pp. 207–223.

[210] V. Forejt, M. Kwiatkowska, and D. Parker, "Pareto curves for probabilistic model checking," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2012, pp. 317–332.

[211] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[212] S. L. Smith, J. Tumová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[213] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Multi-objective search for optimal multi-robot planning with finite LTL specifications and resource constraints," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 768–774.

[214] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239–248, 2016.

[215] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, "Formal approach to the deployment of distributed robotic teams," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 158–171, 2012.

[216] S. Gulwani, O. Polozov, R. Singh *et al.*, "Program synthesis," *Foundations and Trends in Programming Languages*, vol. 4, no. 1-2, pp. 1–119, 2017.