

# **Getting to school on time: Completing linear temporal logic objectives before a fixed deadline with reinforcement learning**

DANIEL BRAUN

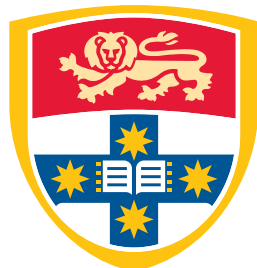
SID: 310270790

Supervisors: Dr Sasha Rubin, Dr Rafael Oliveira

This thesis is submitted in partial fulfillment of  
the requirements for the degree of  
Bachelor of Advanced Studies (Honours) (Computer Science)

School of Computer Science  
The University of Sydney  
Australia

15 November 2021



THE UNIVERSITY OF  
**SYDNEY**

## **Student Plagiarism: Compliance Statement**

I certify that:

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure;

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to the University commencing proceedings against me for potential student misconduct under Chapter 8 of the University of Sydney By-Law 1999 (as amended);

This Work is substantially my own, and to the extent that any part of this Work is not my own I have indicated that it is not my own by Acknowledging the Source of that part or those parts of the Work.

**Name:** Daniel Braun

**Signature:** 

**Date:** 14/11/2021

## Abstract

Objective-based reinforcement learning is concerned with translating an objective specified in a formal language, commonly Linear Temporal Logic, to a numerical reward function on which reinforcement learning algorithms can be applied. In this work, we investigate learning policies that maximise the probability of completing a task before a fixed deadline. This is a common problem setting that reinforcement learning must address if it is to be used widely in real-world applications, but one that has been largely overlooked in the objective-based reinforcement learning literature. We show how methods typically used to “speed up” an agent in RL, such as reward discounting and step-based penalties, are incomplete in this setting. We then introduce QTRM-learning – a novel, temporal-difference-based method that extends the recent work on reward machines to learn optimal policies for this problem. We validate our method experimentally in a slippery gridworld environment, showing that QTRM-learning is much more sample-efficient than alternative methods.

## **Acknowledgements**

I would like to thank my supervisors, Dr Sasha Rubin and Dr Rafael Oliveira, for all the support they have given to me throughout the writing of this dissertation. Without their help in brainstorm interesting research questions, countering and disproving my naive ideas, getting into the weeds with me on tough problems, pointing to relevant literature, and reviewing manuscripts, this thesis output would not have been possible. I am also grateful to my father, Dr Michael Braun, for giving useful comments on earlier drafts of this work, and whose impeccable writing style has helped influence my own writing and thinking.

## CONTENTS

<b>Student Plagiarism: Compliance Statement</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>ix</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
<b>Chapter 2 Background</b>	<b>5</b>
2.1 Markov Decision Process (MDP) .....	5
2.2 Dynamic Programming and Reinforcement Learning .....	9
2.3 Linear Temporal Logic .....	13
2.4 Automata .....	16
<b>Chapter 3 Problem Statement</b>	<b>19</b>
<b>Chapter 4 Related Work</b>	<b>20</b>
4.1 Satisfaction Goals .....	20
4.2 Challenges of Objective Satisfaction in RL .....	21
4.3 Objective-based RL for LTL .....	22
4.3.1 Satisfaction Guarantees without Automata .....	23
4.3.2 Automata-Based Approaches .....	23
4.3.3 Miscellaneous RL Settings for LTL .....	25
4.4 Objective-based RL for Finite LTL .....	26
4.4.1 Direct Translation to Rewards .....	26
4.5 Reward Machines .....	27
4.6 Planning with Finite LTL .....	29

4.7	RL with LTL Landscape .....	29
4.8	Finite Horizon Problem in Reward-Based RL .....	31
4.8.1	Time-Awareness in Deep RL .....	31
<b>Chapter 5</b>	<b>Methods</b>	<b>32</b>
5.1	Objective-Based RL Theorems .....	32
5.2	Objective-Based RL Algorithms .....	36
5.3	Speeding Up the Agent with Heuristics .....	39
<b>Chapter 6</b>	<b>Experimental Evaluation</b>	<b>41</b>
6.1	Frozen Lake Environment .....	41
6.2	Reward Machines for Tasks .....	44
6.3	Optimal Policies .....	44
6.3.1	Infinite Horizon .....	45
6.3.2	Finite Horizon .....	46
6.4	Heuristic Methods .....	49
6.5	Model Efficiency Experiments .....	51
6.6	Code .....	55
<b>Chapter 7</b>	<b>Discussion</b>	<b>56</b>
7.1	Terminal States .....	56
7.2	Other RL Paradigms .....	57
7.3	Improving Update Efficiency .....	57
7.4	Reward Shaping .....	58
7.5	Time-Dependent and Reward-Machine-Dependent Environment Transitions .....	59
7.6	Minimising Time .....	59
<b>Chapter 8</b>	<b>Conclusion</b>	<b>61</b>
	<b>Bibliography</b>	<b>62</b>
	<b>Appendix Appendix</b>	<b>66</b>
1	Additional Background .....	66
2	Additional Figures .....	66

## List of Figures

1.1	Computer games in which RL outperforms humans (Mnih et al. 2015)	1
5.1	MDP with goal state $g$ and initial state sampled uniformly from $\{s_0, s_1\}$ . A reward of 1 is given for transitions to $g$ and 0 otherwise. Edges are labelled with actions and their corresponding transition probabilities.	39
6.1	Frozen Lake environment Task 1: $\varphi_1 = \diamond F$ (i.e. eventually get to Friend)	42
6.2	Frozen Lake environment Task 2: $\varphi_2 = \diamond(R \wedge \diamond F)$ (i.e. eventually get the Rope and then eventually get to Friend).	43
6.3	Frozen Lake environment Task 3: $\varphi_3 = \diamond(R_1 \wedge \diamond(R_2 \wedge \diamond(R_3 \wedge \diamond F)))$ (i.e. eventually get the Ropes in order and then eventually get to Friend).	43
6.4	Simple reward machines for Frozen Lake Tasks 1 and 2. The nodes $u_0, u_1 \in U$ and $b_0 \in B$ represent the non-goal-states and goal states of the RM, respectively (goal states are also marked with a double circle). Edges are labelled by the tuple $(p,r)$ where $p$ is the logical condition governing the transition, and $r$ is the reward corresponding to the transition.	44
6.5	Task 1 optimal infinite horizon values. Cell values indicate the probability of task satisfaction from that state.	45
6.6	The optimal infinite horizon environment state values for each reward machine state in Task 2. Note that the rope is only relevant for $u_0$ (a), as the agent must have already collected the rope to transition to $u_1$ (b).	46
6.7	The optimal finite horizon values obtained from value iteration for Task 1 with $\gamma = 1$ .	48
6.8	The optimal infinite horizon values for Task 1 with different settings for the discount factor $\gamma$ .	49
6.9	Task 1 optimal infinite horizon values with step based penalties of 0.1 and $\gamma = 0.999$ .	50

- 6.10 Infinite horizon sample and update efficiency for Tasks 2 and 3 with  $\gamma = 0.99$ . The black line represents the optimal average reward (obtained using value iteration). Shaded regions represent the average reward of the 25% and 75% quartiles over the 30 independent runs. 52
- 6.11 Finite horizon sample and update efficiency with  $\gamma = 1$  (no discounting). Task 2 has a horizon of 6. Task 3 has a horizon of 15. Shaded regions represent the average reward of the 25% and 75% quartiles over the 30 independent runs. 54
- 2.1 Task 2 reward vs. samples. 67
- 2.2 Task 3 reward vs. samples. 67



## **List of Tables**

- 4.1 Summary of works in objective-based RL addressing the infinite horizon problem. The trace type column indicates whether a finite or infinite trace is needed to verify satisfaction of the formula.

30

## Introduction

---

Reinforcement learning, or RL (Puterman. 1994; Sutton and Barto 2018), lies at the core of several modern breakthroughs in artificial intelligence (Silver, Huang, et al. 2016; Andrychowicz et al. 2020), and continues to be viewed as a promising framework for building agents that can perceive, act, and learn in real-world like humans can (Silver, Singh, et al. 2021). Its success and future promise is due to its natural representation of agent-environment interactions, and its ability to leverage large and efficient computational frameworks such as deep neural networks (Mnih et al. 2015; Schulman et al. 2017).

In the RL setting, an agent takes actions in an environment and receives new observations and numerical rewards in return. The reward function is typically Markovian, i.e. it is defined on state-action pairs, and does not depend on a history of states and actions.<sup>1</sup> The agent does not know, a priori, the transition dynamics of the environment (the probability of moving to a new state given the current state and action), or the reward function, and must learn a policy by interacting with the environment. Figure 1.1 shows two Atari 2600 games in which RL has performed better than human experts.

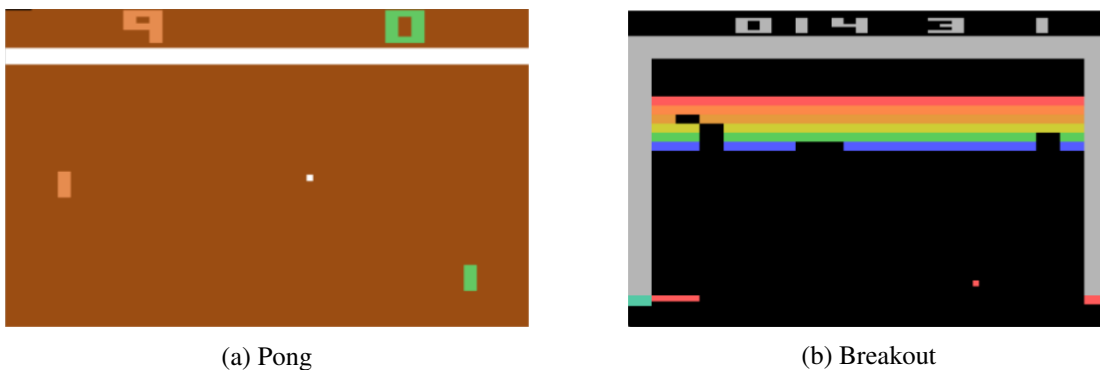


FIGURE 1.1. Computer games in which RL outperforms humans (Mnih et al. 2015)

---

<sup>1</sup>In the RL literature, rewards are also commonly defined on state-action-newstate triples or on individual states, both of which are also Markovian.

In Pong, pictured in Figure 1.1a, the agent gets a positive reward every time it scores a point, and a negative reward every time it concedes a point. In Breakout (Figure 1.1b), the agent gets a positive reward for breaking blocks, and the score is reset if the ball reaches the floor.

There are several factors that are key to RL’s success in these settings that do not come easily in more complex settings. First, Markovian rewards are sufficient for the problem, and are easy to define (e.g. 1 point for any state and action that causes the ball to go off screen in Pong). There are many settings in which it is theoretically impossible to define a Markovian reward function that correctly incentivises the desired behaviour (see Sec 1.1 in Littman et al. 2017). Second, the non-zero rewards are frequent, meaning that the agent can regularly iterate on better policies. Third, the local rewards are well aligned with the global objectives. That is, scoring lots of rewards leads the agent to perform well on the general objectives of the games (e.g. to knock the ball past the opponent without letting it pass you in pong and to break all the blocks without letting the ball hit the floor in Breakout).

This success has inspired work on methods to automatically translate objectives specified in temporal logic languages, the most natural of which being linear temporal logic (LTL), to non-Markovian reward functions for use in RL. Across the RL literature, several techniques have been devised to satisfy objectives specified with LTL formulae, with varying success. We call this problem *objective-based RL*, in contrast to *reward-based RL* where the goal is to maximise rewards from a given numerical reward function. Solutions to the objective-based RL problem generally involve forming a *product MDP* by composing the problem’s Markov Decision Process (MDP) with an automaton representing the LTL formula. Markovian rewards are defined on this product MDP (corresponding to non-Markovian rewards on the original MDP), and the agent uses these rewards to learn a policy with RL that satisfies the objective. Recently, *reward machines* (Icarte et al. 2018b; Icarte et al. 2018a) have been proposed to act as a normal form for representing the non-Markovian reward functions above, allowing task specifications given in different formal languages to be expressed in the same framework.

Commonly overlooked in the literature is a formal consideration of the time taken for an agent to complete an objective expressed with a temporal logic formula. Borrowing from the reward-based RL and planning literature, we call the problem of finding a policy that maximises the probability of completing an objective before a fixed deadline the *finite horizon problem* (in rewards-based RL and planning, the finite horizon problem simply relates to maximising rewards before a deadline). This is in contrast to

the *infinite horizon problem*, in which the agent must maximise the probability of satisfying an objective over an infinite time horizon.<sup>2</sup> There are several reasons that the finite horizon problem has been overlooked in objective-based RL:

- (1) By default, LTL, which can express a very wide variety of objectives, is interpreted on infinite traces. This means that an objective is only deemed to be satisfied when considering an agent’s actions over an infinite number of timesteps. While infinite traces are a necessity for tasks such as “continually monitor rooms A and B”, tasks which have a finite completion time, such as “go to room A without passing room B”, or, indeed, “get to school on time”, do not need infinite traces to verify their satisfaction.
- (2) Adding additional time constraints to objectives increases the complexity of the learning process. As well, methods that are of practical use in RL (e.g. discounted rewards, step-based penalties) often do not align well with objectives that have time considerations (see Sect 5.3 for more details).
- (3) Objective-based RL has yet to see widespread deployment, and has thus not had to deal with the stringent demands of industrial applications, such as completing tasks quickly or within specific time-bounds.

If RL is to be a useful framework for creating agents that perform well on complex everyday tasks, then methods to incorporate time constraints in objective-based RL are paramount. Solutions to this problem will have applications in robotics (e.g. “build this product before the factory closes” or “perform this surgery before the patient bleeds out”), navigation (e.g. “maximise the probability I get to work before the meeting starts”), cyber-security (e.g. “find and eliminate the malware before the systems go back online”), among others. In this vein, our contributions are as follows:

- (1) We critically analyse existing methods in objective-based RL, with a particular focus on how they relate to the finite horizon problem.
- (2) We investigate methods commonly used in the reward-based infinite horizon problem that encourage an agent to “speed up” (i.e. incentivise it to collect rewards more quickly), and show how they lead to sub-optimal policies in the finite horizon setting.
- (3) We introduce QTRM-learning (Q-learning with Time and Reward Machines), a sample-efficient, novel extension of the Q-learning-based methods from the reward machines literature that

---

<sup>2</sup>Note that, in the infinite horizon setting, RL practitioners often train an agent in a series of finite episodes. This is done mostly to ensure that the agent learns to perform well in the initial environment states.

learns policies which maximise the probability of satisfying a temporal logic formula before a fixed deadline.

- (4) We compare experimentally the computational and sample efficiency of QTRM-learning with more primitive algorithms for several tasks in a customised slippery gridworld environment.

We begin in Chapter 2 by introducing the concepts and notation used throughout the thesis. We then present the central problems of the thesis in Chapter 3. In Chapter 4, we review the existing work in the literature related to our problems, before introducing our solutions in Chapter 5. We test the presented methods on various tasks in slippery gridworld experiments in Chapter 6. Further discussion and ideas for future work are laid out in Chapter 7, before concluding in Chapter 8.

## Background

---

In this chapter, we introduce the relevant concepts and notations.

Throughout this work, we use  $\mathbb{N}$  to denote the set  $\{0, 1, 2, \dots\}$ . We denote by  $\mathcal{D}(X)$  the set of all probability distributions on a set  $X \subseteq \mathbb{R}^n$ .

### 2.1 Markov Decision Process (MDP)

**DEFINITION 1 (MDP).** An MDP is a tuple  $M = (S, A, P, \nu_{init}, R, \gamma)$ , where  $S$  is a finite set of states,  $A$  is a finite set of actions,  $P : S \times A \rightarrow \mathcal{D}(S)$  is a function returning a probability distribution over states given a state-action pair (written with the conditional probability notation  $P(s'|s, a)$ ),  $\nu_{init} : S \rightarrow [0, 1]$  is a distribution over starting states of the MDP,  $R : S \times A \times S \rightarrow \mathbb{R}$  is a function which defines a numerical reward for taking an action in a state and arriving in a new state, and  $\gamma \in [0, 1]$  geometrically discounts the rewards given to the agent over time. Let  $A(s)$  represent the set of actions that can be taken in state  $s \in S$ ; then for all  $s \in S$ ,  $\sum_{s' \in S} P(s'|s, a) = 1$  if  $a \in A(s)$ , and 0 otherwise.

We say that an MDP is *deterministic* if the transition function  $P$  maps to single states (i.e.  $P : S \times A \rightarrow S$ ), and *stochastic* otherwise.

**DEFINITION 2 (Labelled MDP).** A labelled MDP is a tuple  $M = (S, A, P, \nu_{init}, R, \gamma, \mathcal{AP}, L)$ , where  $S$ ,  $A$ ,  $P$ ,  $\nu_{init}$ ,  $R$  and  $\gamma$  are as in Definition 1,  $\mathcal{AP}$  is a finite set of atomic propositions, and  $L : S \rightarrow 2^{\mathcal{AP}}$  is a labelling function that maps a state  $s \in S$  to a set of atomic propositions in  $\mathcal{AP}$ .

As we will see in Chapter 5, the labelling function is sometimes instead defined as  $L : S \times A \times S \rightarrow 2^{\mathcal{AP}}$ , mapping  $(s, a, s')$  experiences instead of individual states to a set of atomic propositions.

In this work, we often omit the “labelled” qualifier and just use the term MDP. Also, since we sometimes define reward functions over different objects (e.g. reward machines) than the MDP itself, we commonly define an MDP without a reward function or discount factor (e.g.  $M = (S, A, P, \nu_{\text{init}}, \gamma)$  or  $M = (S, A, P, \nu_{\text{init}}, \mathcal{AP}, L)$ ).

**DEFINITION 3 (Trajectory).** *Given an MDP  $M = (S, A, P, \nu_{\text{init}}, R, \gamma)$ , an infinite trajectory  $\tau$  of  $M$  is a sequence of states and actions  $\tau = (s_0, a_0, s_1, a_1, \dots)$  in  $M$  such that  $\nu_{\text{init}}(s_0) > 0$  and  $P(s_{t+1}|s_t, a_t) > 0$  for all  $t \in \mathbb{N}$ .*

We use  $\tau_t$  to denote the  $t$ th state-action pair  $(s_t, a_t)$  of the trajectory,  $\tau_{t:h}$  to denote the sequence  $(s_t, a_t, \dots, s_h)$ ,  $\tau_t$  to denote  $(s_t, a_t, \dots)$ , and  $\tau_{:t}$  to denote  $(s_0, a_0, \dots, s_t)$ .

In this work, we are also interested in *finite trajectories*. A finite trajectory of length  $H$  has the form  $\tau = (s_0, a_0, \dots, s_{H-1}, a_{H-1}, s_H)$ .<sup>1</sup> We let  $\tau_H$  denote the final state  $s_H$ , as there is no action corresponding to the final timestep. We generally use the same notation for finite and infinite trajectories, although we occasionally use  $\tau^{\text{fin}}$  to specify a finite trajectory. MDPs which emit finite trajectories of a fixed length  $H$  are called *finite horizon MDPs*, whereas MDPs which emit infinite trajectories are called *infinite horizon MDPs*. While infinite horizon MDPs are more common in the RL literature, finite horizon MDPs are a more natural representation to account for maximising rewards within a fixed time frame. Note that it is possible, but often more verbose, to convert finite horizon MDPs to infinite horizon MDPs with the use of *terminal states*: states in the MDP that transition to themselves at every timestep without reward ad infinitum.

The accumulated discounted reward of an infinite trajectory  $\tau = (s_0, a_0, s_1, a_1, \dots)$  in an MDP is given by

$$G(\tau) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}). \quad (2.1)$$

For a finite trajectory of length  $H$ , the summation is instead from 0 to  $H - 1$ .

**DEFINITION 4 (Trace).** *Given a trajectory  $\tau$  in a labelled MDP  $M = (S, A, P, \nu_{\text{init}}, R, \gamma, \mathcal{AP}, L)$ , a trace  $tr(\tau)$  on  $\tau$  is the sequence of labelled states in  $\tau$ . That is,*

$$tr(\tau) = (L(s_t))_{s_t \in \tau}.$$

<sup>1</sup>The “length” corresponds to the number of state-action pairs, not the number of states. This is convenient later in order to say that a trajectory of length  $H$  can satisfy an objective on the  $H_{\text{th}}$  timestep

**DEFINITION 5 (Policy).** A memoryless policy (also known as a stationary policy or positional policy)  $\pi : S \rightarrow \mathcal{D}(A)$  on an MDP is a function mapping states to a probability distribution over actions. A time-dependent policy (also known as a non-stationary policy)  $\pi : S \times T \rightarrow \mathcal{D}(A)$  maps a state and the current timestep to a distribution over actions. A history-dependent policy  $\pi : (S \times A)^* \times S \rightarrow \mathcal{D}(A)$  is a policy dependent on finite trajectories (i.e. a finite number of state-action pairs and the current state).

We often use the conditional probability notation  $\pi(a|s)$  to represent  $\pi(s)(a)$  for memoryless policies,  $\pi(a|s, t)$  to represent  $\pi(s, t)(a)$  for time-dependent policies, and  $\pi(a|\tau:t)$  to represent  $\pi(\tau:t)(a)$  for history-dependent policies. Note that history-dependent policies are a superset of both memoryless and time-dependent policies, and we may thus use the  $\pi(a|\tau:t)$  notation regardless of the type of policy.

**DEFINITION 6 (Trajectories Induced by a Policy).** Given an MDP  $M = (S, A, P, \iota_{init}, R, \gamma)$ . The set of all infinite trajectories of the form  $\tau = (s_0, a_0, s_1, a_1, \dots)$  induced by a policy  $\pi$  is given by:

$$\text{Traj}_{\pi, M} := \{\tau : \iota_{init}(s_0) > 0, \pi(a_t|\tau:t) > 0 \text{ and } P(s_{t+1}|s_t, a_t) > 0 \text{ for all } t \in [0, \infty)\}.$$

The set of all finite trajectories of the form  $\tau:h = (s_0, a_0, s_1, a_1, \dots, s_h)$  induced by a policy  $\pi$  is given by:

$$\text{Traj}_{\pi, M}^{fin} := \{\tau:h : h \in \mathbb{N}, \iota_{init}(s_0) > 0, \pi(a_t|\tau:t) > 0 \text{ and } P(s_{t+1}|s_t, a_t) > 0 \text{ for all } t \in [0, h-1]\}.$$

The set of all finite trajectories of fixed length  $H$  induced by a policy  $\pi$  is simply  $\text{Traj}_{\pi, M}^H := \{\tau^{fin} \in \text{Traj}_{\pi, M}^{fin} : |\tau^{fin}| = H\}$ . For brevity, we omit the  $M$  and simply write  $\text{Traj}_{\pi}$ ,  $\text{Traj}_{\pi}^{fin}$  and  $\text{Traj}_{\pi}^H$ , as the MDP  $M$  is generally obvious from the context.

Much of this work requires reasoning about properties of subsets of  $\text{Traj}_{\pi}^H$  and  $\text{Traj}_{\pi}$ . In order to formally define a notion of probabilities over sets of infinite trajectories  $\text{Traj}_{\pi}$ , we must first introduce *cylinder sets*.

**DEFINITION 7 (Cylinder Set).** Given an MDP  $M = (S, A, P, \iota_{init}, R, \gamma)$ , the cylinder set of a finite trajectory  $\tau^{fin} = (s_0, a_0, \dots, s_h) \in \text{Traj}_{\pi}^{fin}$  on  $M$  induced by a policy  $\pi$  is given by:

$$\text{Cyl}_{\pi}(\tau^{fin}) = \{\tau \in \text{Traj}_{\pi} : \tau^{fin} \in \text{pref}(\tau)\}$$

where  $\text{pref}(\tau)$  represents the set of all finite prefixes of the infinite trajectory  $\tau$ . Informally,  $\text{Cyl}_{\pi}(\tau^{fin})$  is the set of all infinite trajectories that start with  $\tau^{fin}$  and follow the policy  $\pi$ .



Using cylinder sets, we can now formally define the probability space over infinite trajectories under a policy.

**DEFINITION 8 (Probability Space over Infinite Trajectories).** *Given a policy  $\pi$  on an MDP  $M = (S, A, P, \iota_{init}, R, \gamma)$ , the probability space over infinite trajectories on  $M$  induced by  $\pi$  is given by the tuple  $(\text{Traj}_\pi, \mathcal{E}_\pi, \text{Pr}_\pi^{inf})$ , where:*

- (1)  $\text{Traj}_\pi$  is the set of all infinite trajectories in  $M$  induced by  $\pi$ , as defined in Definition 6.
- (2)  $\mathcal{E}_\pi$  is the smallest  $\sigma$ -algebra that contains the cylinder sets  $\text{Cyl}_\pi(\tau)$  for all  $\tau \in \text{Traj}_\pi^{fin}$ . I.e. the set of all cylinders induced by  $\pi$ . See Definition 18 in the Appendix for a formal definition of a  $\sigma$ -algebra.
- (3)  $\text{Pr}_\pi^{inf} : \mathcal{E}_\pi \rightarrow [0, 1]$  is the probability measure defined canonically on cylinders produced by finite trajectories of the form  $\tau_{:h} = (s_0, a_1, \dots, s_h)$ :

$$\text{Pr}_\pi^{inf}(\text{Cyl}_\pi(\tau_{:h})) = \iota_{init}(s_0) \prod_{t=0}^{h-1} \pi(a_t | \tau_{:t}) P(s_{t+1} | s_t, a_t).$$

We use  $\mathbb{E}_\pi^{inf}$  to denote the expectation operator for this probability space.

For the case of finite trajectories of length  $H$ , we can simply define the probability space as follows:

**DEFINITION 9 (Probability Space over Finite Trajectories).** *Given a policy  $\pi$  on an MDP  $M = (S, A, P, \iota_{init}, R, \gamma)$ , the probability space over finite trajectories of length  $H$  on  $M$  induced by  $\pi$  is given by the tuple  $(\text{Traj}_\pi^H, \mathcal{E}_\pi, \text{Pr}_\pi^H)$ , where:*

- (1)  $\text{Traj}_\pi^H$  is the set of all finite trajectories of length  $H$  in  $M$  induced by  $\pi$ , as defined in Definition 6.
- (2)  $\mathcal{E}_\pi = 2^{\text{Traj}_\pi^H}$ , that is, the set of all subsets of  $\text{Traj}_\pi^H$ , including the empty set and the set  $\text{Traj}_\pi^H$  itself.
- (3)  $\text{Pr}_\pi^H : \mathcal{E}_\pi \rightarrow [0, 1]$  is the probability measure defined canonically on finite trajectories of length  $H$ :

$$\text{Pr}_\pi^H(\tau_{:H}) = \iota_{init}(s_0) \prod_{t=0}^{H-1} \pi(a_t | \tau_{:t}) P(s_{t+1} | s_t, a_t).$$

We use  $\mathbb{E}_\pi^H$  to denote the expectation operator for this probability space.

Now that we have defined probability spaces over trajectories, we can define what it means for a policy to be *optimal* in an MDP.

**DEFINITION 10 (Optimal Policy).** *A policy is optimal if it maximises the expected future discounted reward. For an infinite horizon problem, an optimal policy is given by:*

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{\pi}^{\text{inf}}[G(\tau)].$$

*Similarly, the optimal policy in a finite horizon problem with deadline  $H$  is given by:*

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{\pi}^H[G(\tau:H)].$$

While we define optimal policies over trajectories with starting states governed by  $\iota_{\text{init}}$ , it is common to define an optimal policy to be one which maximises the expected future discounted reward from any starting state in the MDP (Sutton and Barto 2018). These two definitions are equivalent when  $\iota_{\text{init}}(s) > 0$  for all  $s \in S$ , but the latter definition is more stringent when this condition does not hold. Nonetheless, the methods discussed in this paper adhere to both definitions of optimality.

## 2.2 Dynamic Programming and Reinforcement Learning

In the reinforcement learning context, neither the transition function  $P$  nor the reward function  $R$  are known to the agent a priori, and the agent must learn the optimal policy by gathering information from the environment (and thus sampling from these functions). Some RL methods construct a transition and reward function, and use them to solve the subsequent planning problem (*model-based RL*), while others learn a policy directly without a model of the environment (*model-free RL*). For this research, we look at solutions to our problem using model-free RL due to its scalability to large applications (Strehl et al. 2006).

When the transition and reward functions are known, we can compute optimal policies of an MDP using techniques from *dynamic programming* (Bellman 1966). These techniques form the basis of reinforcement learning, and are introduced below. In the following, we use  $S_h$ ,  $A_h$  and  $R_h$  to denote the state, action and reward at timestep  $h$ , respectively, where the reward at timestep  $h$  depends on the action and state at timestep  $h - 1$ . These are random variables over the probability space induced by a policy.

Given a horizon of  $H$ , and discount factor  $\lambda \in [0, 1]$  the expected future discounted reward from each state and each timestep  $h \in \{0, 1, \dots, H - 1\}$  when following policy  $\pi$  is given by the *value function*:

$$V_h^\pi(s) = \mathbb{E}_\pi^H \left[ \sum_{t=h+1}^H \gamma^{t-(h+1)} R_t | S_h = s \right].$$

We define  $V_H^\pi(s) = 0$ , as no future rewards can be obtained on the final timestep  $H$ . The *Q-function*, also known as the *action-value function*, gives the value of taking an action in a state starting at timestep  $h \in \{0, 1, \dots, H - 1\}$  and following a policy  $\pi$  thereafter:

$$Q_h^\pi(s, a) = \mathbb{E}_\pi^H \left[ \sum_{t=h+1}^H \gamma^{t-(h+1)} R_t | S_h = s, A_h = a \right].$$

As with the value function, we define  $Q_H^\pi(s, a) = 0$  for all  $a \in A(s)$ . Note that the value of a state  $s$  at timestep  $H - 1$  is simply the expected reward at the next (and final) timestep when following  $\pi$ . I.e.  $V_{H-1}^\pi(s) = \mathbb{E}_\pi^H [R_H | S_{H-1} = s]$ . Similarly, the Q-value of a state  $s$  and action  $a$  at timestep  $H - 1$  is  $Q_{H-1}^\pi(s, a) = \mathbb{E} [R_H | S_{H-1} = s, A_{H-1} = a]$ , and is in fact independent of the policy  $\pi$ .

When  $\pi$  is an optimal policy, denoted  $\pi^*$  or just  $*$ , it satisfies the Bellman optimality equations (Bellman 1966):

$$V_h^*(s) = \max_a \mathbb{E} [R_{h+1} + \gamma V_{h+1}^*(S_{h+1}) | S_h = s, A_h = a] \quad (2.2)$$

$$= \max_a \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma V_{h+1}^*(s')], \quad (2.3)$$

and

$$Q_h^*(s, a) = \mathbb{E} [R_{h+1} + \gamma V_{h+1}^*(S_{h+1}) | S_h = s, A_h = a] \quad (2.4)$$

$$= \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma V_{h+1}^*(s')] \quad (2.5)$$

$$= \sum_{s'} P(s' | s, a) [R(s, a, s') + \gamma \max_{a'} Q_{h+1}^*(s', a')] \quad (2.6)$$

for all states  $s \in S$ , timesteps  $h \in \{0, 1, \dots, H - 1\}$  and  $\lambda \in [0, 1]$ . Note that the last equality comes from the fact that the optimal value at a state is equivalent to the maximum Q-value at that state. For the infinite horizon setting, we have the equivalent equations with values  $V^*$  and action values  $Q^*$  that are

independent of the timestep:

$$\begin{aligned} V^*(s) &= \max_a \mathbb{E}[R_{h+1} + \gamma V^*(S_{h+1}) | S_h = s, A_h = a] \\ &= \max_a \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V^*(s')], \end{aligned}$$

and

$$\begin{aligned} Q^*(s, a) &= \mathbb{E}[R_{h+1} + \gamma V^*(S_{h+1}) | S_h = s, A_h = a] \\ &= \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V^*(s')] \\ &= \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma \max_{a'} Q^*(s', a')]. \end{aligned}$$

The existence and uniqueness of  $V^*$  and  $Q^*$  are guaranteed provided either  $\gamma < 1$  or there is some point at which no further rewards are received.

As described in Sutton and Barto 2018, we can obtain the optimal value function in an MDP using an iterative algorithm called *value iteration*. The finite horizon version is shown in Algorithm 1, and the infinite horizon version is shown in Algorithm 2.

---

**Algorithm 1** Finite Horizon Value Iteration

---

**Input:** MDP  $M = (S, A, P, R, \gamma)$  with  $\gamma \in [0, 1]$ , horizon  $H$   
 $V_h(s) \leftarrow 0$  for all  $s \in S$  and  $h \in \{H, H - 1, \dots, 0\}$   
**for**  $h \in \{H - 1, H - 2, \dots, 0\}$  **do**  
    **for**  $s \in S$  **do**  
         $V_h(s) \leftarrow \max_a \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V_{h+1}(s')]$   
    **end for**  
**end for**  
**Output:** Optimal values  $V$

---

Note that, in the finite horizon version, we iterate through the timesteps backwards from  $H - 1$  to 0 so that we only have to do one pass through the set of timesteps as opposed to  $H$  passes. In the infinite horizon case, we update the value function until the changes in values for one iteration are all below a particular threshold. In these algorithms, it is also possible to initialise the values arbitrarily, provided there is a notion of a terminal state(s) (in which no future rewards can be made) whose value(s) is initialised to 0.

For horizon  $H$  and discount factor  $\gamma \in [0, 1]$ , we can obtain an optimal deterministic time-dependent policy for each state  $s \in S$  and timestep  $h \in \{0, 1, \dots, H - 1\}$  by choosing an action that maximises the

---

**Algorithm 2** Infinite Horizon Value Iteration

---

**Input:** MDP  $M = (S, A, P, R, \gamma)$  with  $\gamma \in [0, 1]$ , small threshold  $\theta > 0$  $V(s) \leftarrow 0$  for all  $s \in S$ **repeat** $\Delta \leftarrow 0$ **for**  $s \in S$  **do** $v \leftarrow V(s)$  $V(s) \leftarrow \max_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V(s')]$  $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ **end for****until**  $\Delta < \theta$ **Output:** Optimal values  $V$ 

---

reward and pre-computed optimal value function at the following timestep:

$$\pi^*(s, h) = \operatorname{argmax}_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V_{h+1}^*(s')]. \quad (2.7)$$

For the infinite horizon setting, we similarly have  $\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')]$ , but this is only true for  $\gamma \in [0, 1)$  (Sutton and Barto 2018). For an intuitive example of why this does not hold for  $\gamma = 1$ , see Section 6.3.1. This difference is not trivial for objective-based RL, as we need  $\gamma = 1$  in order to find a policy that maximises the satisfaction probability of an objective. For finding an optimal policy in the infinite horizon setting with  $\gamma = 1$ , instead of choosing a single action that maximises the immediate reward and optimal value at the next timestep, one can form an optimal stochastic policy by assigning a non-zero probability to each action that maximises the immediate reward and optimal value at the next timestep. That is,  $\pi^*(s)(a) > 0$  for all  $a \in \operatorname{argmax}_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')]$ . This prevents the agent from getting stuck in an infinite loop without receiving rewards when taking deterministic actions amongst states that have optimal values.

Q-learning (Watkins and Dayan 1992) is a method for learning the optimal Q-functions without the agent having access to the transition function  $P$  and reward function  $R$  (i.e. model-free RL). In the finite horizon case (Algorithm 3), it works by initialising the Q-values for each timestep and each state-action pair, taking an action in the environment, observing a new state and reward, and updating the Q-value using the observed reward and the Q-value of the successor state. The Q-values are known to converge to the optimal Q-values provided each state-action pair is visited infinitely often (Puterman. 1994; Harada 1997). In practice, the algorithm is run for fixed number of *episodes*. An episode starts at an initial state with the timestep and reward counters set to 0. After a pre-determined number of states, or when the agent reaches a terminal state, the state, timestep counter and rewards get re-initialised and the agent

begins again. Q-learning is an *off-policy* algorithm in the sense that it can operate entirely on sampled experiences that are independent of any particular policy. This is in contrast to on-policy algorithms, that require the sampled experiences to be related to the current policy at that point in training. However, Q-learning is typically implemented in an  $\epsilon$ -greedy manner, sampling random actions  $\epsilon$  percent of the time and following a greedy policy (i.e. picking the actions with the highest currently estimated Q-value) the rest of the time. Q-learning is known as a *temporal difference* learning method, as it learns to predict a quantity (the Q-value of the current state) using a future signal (the q-values of the subsequent state).

---

**Algorithm 3** Finite Horizon Q-learning
 

---

**Input:** MDP  $M = (S, A, P, \iota_{\text{init}}, R, \gamma)$  with  $\gamma \in [0, 1]$ , horizon  $H$ , learning rate  $\alpha \in (0, 1]$ ,  $\epsilon \in (0, 1]$   
 For all  $s \in S$ ,  $a \in A$ , initialise  $Q_h(s, a)$  arbitrarily for all  $h \in \{0, 1, \dots, H - 1\}$  and  $Q_H(s, a) \leftarrow 0$   
**for** each episode until termination condition **do**  
   Sample starting state  $s$  from  $\iota_{\text{init}}$   
   **for** each step in episode  $h \in \{0, 1, \dots, H - 1\}$  **do**  
     Take action  $a$  using  $\epsilon$ -greedy policy on  $Q_h$  in current state  $s$ , observing  $s'$   
      $Q_h(s, a) \leftarrow Q_h(s, a) + \alpha[R(s, a, s') + \gamma \max_a Q_{h+1}(s', a) - Q_h(s, a)]$   
     Update  $s \leftarrow s'$   
   **end for**  
**end for**  
**Output:** Optimal action values  $Q$

---

As is the case for value iteration, the Q-learning algorithm for infinite horizon MDPs is the same as for finite horizon MDPs but with all timesteps parameters ignored (Sutton and Barto 2018, Sec 6.5).

While Q-learning is only one of many model-free reinforcement learning algorithms (Sutton and Barto 2018; Y. Li 2017), we use it in this work due to its simplicity, its extensibility to algorithms that handle automata-based rewards (Section 5), and its ability to extend to complex environments that require function approximation (Mnih et al. 2015).

## 2.3 Linear Temporal Logic

In this section, we recall Linear Temporal Logic (LTL) and present its semantics on infinite traces. We then recall variations of LTL, namely  $\text{LTL}_f$  and co-safe LTL, that can be interpreted over finite traces and are thus suited to finite horizon problems.

LTL was first introduced in Pnueli 1977. Its goal was to act as an expressive logic for the specification of non-terminating programs. LTL formulae are built from a set of atomic propositions  $\mathcal{AP}$ , combined with the propositional logic operators *negation* ( $\neg$ ) and *conjunction* ( $\wedge$ ), and temporal operators *next*

( $\bigcirc$ ) and *until* ( $\mathcal{U}$ ). Every proposition  $p \in \mathcal{AP}$  is considered a formula (including the proposition *true*), and if  $\varphi$  and  $\phi$  are formulae, then the following are also considered formulae

$$\neg\varphi, \varphi \wedge \phi, \bigcirc\varphi, \varphi\mathcal{U}\phi.$$

In a labelled MDP, an LTL formula expresses objectives that are interpreted over infinite traces of words in  $2^{\mathcal{AP}}$ , where  $\mathcal{AP}$  is the set of atomic proposition in the labelled MDP. Recall that the labelling function maps a state in the MDP to a set of atomic propositions in  $\mathcal{AP}$ , and an infinite trace is the infinite sequence of labelled states generated by an agent's trajectory (i.e. its sequence of visited states in the environment). LTL allows for the expression of a wide range of objectives. It is capable of expressing *continuous tasks* (also known as *infinite tasks*) that do not terminate in finite time, such as “continually monitor the room temperature”, as well as *finite tasks* that do terminate in finite time, such as “first walk to room A, then walk to room B, all the while avoiding passing room C”. The semantics of LTL formulae are given in Definition 11.

**DEFINITION 11 (LTL Semantics on Infinite Traces).** *Given an LTL formula  $\varphi$  and an infinite trajectory  $\tau = (s_0, a_0, s_1, a_1, \dots)$  in a labelled MDP  $M = (S, A, P, \iota_{init}, \mathcal{AP}, L)$ , we inductively define when  $\varphi$  is satisfied at timestep  $t \in \mathbb{N}$ , written  $\tau, t \models \varphi$ , by:*

$$\begin{aligned} \tau, t \models \text{true} & \text{ for all } t, \\ \tau, t \models p \in \mathcal{AP} & \Leftrightarrow p \in L(s_t), \\ \tau, t \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow (\tau, t \models \varphi_1) \text{ and } (\tau, t \models \varphi_2), \\ \tau, t \models \neg\varphi & \Leftrightarrow \tau, t \not\models \varphi, \\ \tau, t \models \bigcirc\varphi & \Leftrightarrow \tau, t+1 \models \varphi, \\ \tau, t \models \varphi_1\mathcal{U}\varphi_2 & \Leftrightarrow \exists j \text{ s.t. } t \leq j \text{ and } \tau, j \models \varphi_2, \\ & \text{and } \forall i \text{ s.t. } t \leq i < j, \text{ we have } \tau, i \models \varphi_1. \end{aligned}$$

*An infinite trajectory  $\tau$  satisfies an LTL formula  $\varphi$ , written  $\tau \models \varphi$ , if and only if  $\tau, 0 \models \varphi$ .*

In words: An atomic proposition is true at timestep  $t$  in a trajectory if and only if that atomic proposition is one of the labels of the state at timestep  $t$ ; the conjunction of formulae are true at  $t$  if and only if each formulae are true at  $t$ ; the next operator  $\bigcirc$  will cause a formula to be true if and only if it is true at the next timestep; the until operator  $\mathcal{U}$  enforces the formula before the operator to be true until

the formula after the operator is true. Useful extensions of these operators are the *eventually* operator  $\diamond\varphi := \text{true}\mathcal{U}\varphi$ , indicating that  $\varphi$  will eventually become true, and the *always* operator  $\Box\varphi := \neg\diamond\neg\varphi$ , indicating that  $\varphi$  is always true.

The expressiveness of LTL has led to its use in the vast majority of work in objective-based RL, much of which is detailed in Chapter 4. Methods developed to handle any objective expressible with LTL overlook the value that can be obtained by restricting the set of objectives expressible to those that can be completed in finite time. In particular, finite tasks allow for the consideration of the time taken for an agent to complete its objective. This notion is nonsensical for continuous tasks such as “continually monitor the room temperature”. As a result, existing solutions for RL with LTL objectives, the most common objective-based RL setting, do not consider time-costs.

Fortunately, it is possible to interpret LTL on finite traces while keeping the same syntax of LTL on infinite traces. This language is often called *LTL over finite traces* ( $\text{LTL}_f$ ) (Wilke 1999; Gabaldon 2004; Bienvenu et al. 2006). It is shown in Sistla and Clarke 1985 that LTL and  $\text{LTL}_f$  both have similar computational characteristics; satisfiability, validity, and logical implication are P-SPACE complete.

The semantics of  $\text{LTL}_f$  are defined in much the same way as LTL. However, since the trajectories are finite, they are interpreted on timesteps  $t \in \{0, 1, \dots, |\tau|\}$  where  $|\tau|$  is the length of the trajectory. We also have the following minor modifications to account for finite trajectories:

$$\begin{aligned} \tau, t \models \bigcirc\varphi &\Leftrightarrow t < |\tau| \text{ and } \tau, t+1 \models \varphi, \\ \tau, t \models \varphi_1\mathcal{U}\varphi_2 &\Leftrightarrow \exists j \text{ s.t. } t \leq j \leq |\tau| \text{ and } \tau, j \models \varphi_2, \\ &\text{and } \forall i \text{ s.t. } t \leq i < j, \text{ we have } \tau, i \models \varphi_1. \end{aligned}$$

In particular, the next operator applied to a formula  $\bigcirc\varphi$  is false on the final timestep  $|\tau|$ . As in the infinite case, we say a finite trajectory  $\tau$  satisfies an  $\text{LTL}_f$  formula  $\varphi$ , written  $\tau \models \varphi$ , if and only if  $\tau, 0 \models \varphi$ .

Other variations of LTL that handle finite traces have also been formulated. Giacomo and Vardi 2013 introduces *linear dynamic logic on finite traces* ( $\text{LDL}_f$ ), and shows that it shares the computational characteristics with  $\text{LTL}_f$ , and is strictly more expressive.

Another LTL variant of interest in this work is that of co-safe LTL, introduced in Kupferman and Vardi 2001. Despite being interpreted on infinite traces, co-safe LTL can only express finite tasks. This is



due to the fact that each trace that satisfies a co-safe LTL formula contains a finite prefix which, when extended with any suffix, will satisfy the formula. For example, a co-safe LTL objective of “go to room A” will be satisfied for any infinite trace corresponding to an agent visiting room A, regardless of what it does afterwards. More formally, co-safe LTL differs from LTL and  $LTL_f$  in that the negation operator ( $\neg$ ) can only be applied to atomic propositions, and not formulae. This means that co-safe LTL cannot express the always operator  $\Box\varphi := \neg\Diamond\neg\varphi$  and formulae such as  $\neg\Diamond\varphi$ , since it cannot ensure that all suffixes of a good finite prefix will satisfy the formula.

Nonetheless, co-safe LTL,  $LTL_f$ , and  $LDL_f$  are all able to express a wide variety of finite tasks, and we explore methods that apply to all of these logics.

We use the term *finite LTL* formulas to represent a formula in co-safe LTL,  $LTL_f$ , or  $LDL_f$ , as a finite trace is enough to verify their satisfaction.

DEFINITION 12 (Finite LTL Satisfaction Probability). *The probability of a policy  $\pi$  satisfying a finite LTL formula  $\varphi$  in an MDP by the deadline  $H$  is given by:*

$$\Pr_{\pi}^H(\varphi) := \Pr_{\pi}^H(\{\tau \in \text{Traj}_{\pi}^H : \tau \models \varphi\}).$$

## 2.4 Automata

In order to track and verify the satisfaction of a finite LTL formula, we leverage the fact that one can construct automata that accept precisely the computations satisfied by the formula (Kupferman and Vardi 2001; J. A. Baier and McIlraith 2006). In particular, one can construct a *nondeterministic finite automaton* (NFA) (Definition 13) which accepts all and only the runs that satisfy a given formula. This NFA can then be converted to a *deterministic finite automaton* (DFA) (Definition 14) using *powerset construction* (Rabin and Scott 1959).

DEFINITION 13 (Nondeterministic Finite Automaton). *A nondeterministic finite automaton (NFA) is a tuple  $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ , where  $\Sigma$  is a finite nonempty alphabet,  $Q$  is a finite nonempty set of states,  $q_0 \in Q$  is an initial state,  $\delta : Q \times \Sigma \rightarrow 2^Q$  is a transition function, and  $F \subseteq Q$  is a set of accepting states. Intuitively,  $\delta(q, a)$  returns the set of states that  $\mathcal{A}$  can transition to from state  $q \in Q$  after reading the symbol  $a \in \Sigma$ .*

DEFINITION 14 (Deterministic Finite Automaton). *A deterministic finite automaton (DFA) is an NFA with  $|Q_0| = 1$  and  $|\delta(q, a)| \leq 1$  for all states  $q \in Q$  and symbols  $a \in \Sigma$ . That is, there is only a single initial state, and the subsequent state of each transition is entirely determined.*

Notice that, in NFAs and DFAs, there is no output associated with each transition. As a result, methods that use these automata in objective-based RL to track the status of an objective must define a reward function separately.

A recently developed automaton-based framework that natively supports rewards is a *Reward Machine*. First introduced in Icarte et al. 2018b, and most rigorously explored in Icarte et al. 2020, *reward machines* (RMs) (Definition 15) are a type of finite automaton that can express objectives from any logic that can be expressed as a DFA (including LTL<sub>f</sub>, LDL<sub>f</sub>, and co-safe LTL), and pass corresponding non-Markovian rewards to an RL agent.

DEFINITION 15 (Reward Machine). *Given a labelled MDP  $M = (S, A, P, \iota_{init}, \gamma, \mathcal{AP}, L)$  with  $L : S \times A \times S \rightarrow 2^{\mathcal{AP}}$ , a reward machine is a tuple  $\mathcal{R}_M = (U, u_0, B, \delta_u, \delta_r)$ , where  $U$  is a finite nonempty set of states,  $u_0 \in U$  is an initial state,  $B$  is a finite set of terminal states with  $U \cap B = \emptyset$ ,  $\delta_u : U \times 2^{\mathcal{AP}} \rightarrow U \cup B$  is the state-transition function, and  $\delta_r : U \rightarrow [S \times A \times S \rightarrow \mathbb{R}]$  is the state-reward function. A simple reward machine is a reward machine whose state-reward function returns a real number instead of a function. That is,  $\delta_r : U \times 2^{\mathcal{AP}} \rightarrow \mathbb{R}$ .*

DEFINITION 16 (Markov Decision Process with a Reward Machine). *A Markov decision process with a reward machine (MDPRM) is a tuple  $\mathcal{T} = (S, A, P, \iota_{init}, \gamma, \mathcal{AP}, L, U, u_0, B, \delta_u, \delta_r)$ , where  $S, A, P, \iota_{init}, \gamma, \mathcal{AP}$  and  $L : S \times A \times S \rightarrow 2^{\mathcal{AP}}$  are defined as in an MDP, and  $U, u_0, B, \delta_u, \delta_r$  are defined as in an RM.*

A policy in an MDPRM is a function  $\pi : S \times U \rightarrow \mathcal{D}(A(s))$ . In fact, we can view an MDPRM as an MDP with extended state space  $S \times (U \cup B)$  using the transformation in Definition 17.

DEFINITION 17 (Reward Machine Product Markov Decision Process). *Given an MDPRM  $\mathcal{T} = (S, A, P, \iota_{init}, \gamma, \mathcal{AP}, L, U, u_0, B, \delta_u, \delta_r)$ , an RMxMDP is the tuple  $M_{\mathcal{T}} = (S', A', P', \iota'_{init}, R', \gamma')$  where  $S' = S \times (U \cup B)$ ,  $A' = A$ ,  $\gamma' = \gamma$ ,  $\iota'_{init} : S' \rightarrow [0, 1]$  with  $\iota'_{init}(s, u_0) = \iota_{init}(s)$  for all  $s \in S$ ,*

$$P'((s', u')|(s, u), a) = \begin{cases} P(s'|s, a) & \text{if } u' = \delta_u(u, L(s, a, s')) \text{ or } (u \in B \text{ and } u' = u) \\ 0 & \text{otherwise,} \end{cases}$$

and  $R'((s, u), a, (s', u')) = \delta_r(u, L(s, a, s'))$ .

Note that any policy  $\mathcal{T}$  achieves the same reward in  $M_{\mathcal{T}}$ , and vice versa. We make use of this observation to prove properties of reward machines in Section 5.

Camacho, Icarte, et al. 2019 shows how to transform any DFA to a reward machine. As we will see in Chapter 5, this transformation allows one to learn policies that maximise the satisfaction probability of a temporal logic formula by using RL with rewards supplied by a reward machine.

## Problem Statement

---

PROBLEM 1. Suppose we have an MDP  $M = (S, A, P, \iota_{init}, \mathcal{AP}, L)$  and an objective expressed by a finite LTL formula  $\varphi$ . Find a policy which maximises the probability of satisfying  $\varphi$  in the infinite horizon setting. That is, find  $\pi^*$  such that:

$$\pi^* = \operatorname{argmax}_{\pi} Pr_{\pi}(\varphi).$$

Moreover, show that one can learn such a policy using reinforcement learning.

Problem 1 is a version of the infinite horizon problem (since the goal is to maximise the probability of satisfying a formula with no deadline) applied to finite LTL formulae (whose satisfaction can be determined after a finite number of timesteps). As we will see in Section 4.4, solutions to this problem have been proposed in the objective-based RL literature. We investigate the proposed solutions that use the reward machine framework, and empirically analyse various properties of these solutions in new environments.

PROBLEM 2. Suppose we have an MDP  $M = (S, A, P, \iota_{init}, \mathcal{AP}, L)$  and an objective expressed by a finite LTL formula  $\varphi$ . Find a policy which maximises the probability of satisfying  $\varphi$  by the deadline  $H$ . That is, find  $\pi^*$  such that:

$$\pi^* = \operatorname{argmax}_{\pi} Pr_{\pi}^H(\varphi).$$

Additionally, show that such a policy can be learned in the reinforcement learning setting.

In Chapter 5, we propose a solution to the finite horizon Problem 2. Using the reward machines framework, we introduce a highly sample-efficient new algorithm called QTRM-learning that solve this problem in the RL setting, before testing this and similar algorithms experimentally in Chapter 6.

## Related Work

---

In this chapter, we analyse existing methods that address the infinite horizon problem in objective-based RL, both in the finite trace (finite LTL) and infinite trace (LTL) settings. We also examine methods that deal with the finite horizon problem in the RL setting where the numerical reward function is given (instead of a temporal logic objective). We then combine methods from both of these areas to handle the finite horizon problem in Chapter 5.

### 4.1 Satisfaction Goals

In a stochastic MDP, taking a particular action results in the state transitioning probabilistically to one of a set of new states. Compared to deterministic MDPs, in which the future state depends only on the current state and action, stochastic MDPs better suit settings in which it is impossible to model every aspect of the environment deterministically. For this reason, stochastic MDPs have attracted increasing attention in the RL and planning literature. In stochastic environments, a single policy may result in many different traces, some of which may or may not satisfy a temporal logic formula. Therefore, in addition to specifying a formula, the problem designer must decide what satisfaction goals they would like the agent to achieve. The objective-based RL literature has so far focused on two distinct satisfaction goals:

**G1** Find a policy that satisfies a temporal logic formula with probability 1, if such a policy exists.

**G2** Find a policy that maximises the probability of satisfying a temporal logic formula.

**G1** was the focus of Hasanbeig, Abate, et al. 2018; Oura et al. 2020; Wang et al. 2020. This rigid goal is suitable for the many cases where there must be guarantees that an agent will complete its task. For example, a surgical robot may be tasked with performing laser eye surgery on a patient without causing injury. If an RL agent learning this task (in simulations!) fails to find a strategy that always avoids

harming the patient, then, assuming humans can always perform the task without causing injury, the robot should not be used to perform this task. The obvious problem with this satisfaction goal, is that in most cases, it is impossible to satisfy the goal with probability 1. In the articles cited above (Hasanbeig, Abate, et al. 2018; Oura et al. 2020; Wang et al. 2020) that develop methods for this satisfaction goal, it is often shown empirically that agents tasked with this rigid satisfaction goal generally learn appropriate policies even if they cannot ensure satisfaction. However, these works do not make formal guarantees about the satisfaction probability of learned policies.

The more general satisfaction goal **G2** was examined in Hasanbeig, Kantaros, et al. 2019; Hasanbeig, Abate, et al. 2020a; Littman et al. 2017; A K Bozkurt et al. 2020; Alper Kamil Bozkurt et al. 2021; Hahn et al. 2019. In these works, the authors were able to create an MDP such that the policy which maximised rewards in the MDP, also maximised the probability of satisfying the formula. Note that this goal subsumes **G1**, in that it will also find any policy that satisfies a formula with probability 1, if one exists. Work in this direction is a promising step towards methods that give the designer more control over task specification, and is thus the goal we attempt to solve in this work.

## 4.2 Challenges of Objective Satisfaction in RL

There are two notable challenges in creating methods that allow learning policies satisfying temporal logic objectives in RL:

- (1) *sparse rewards*;
- (2) the Simulation Lemma (Kearns and Singh 2002) does not hold for LTL on infinite traces.

The default approach to defining a reward function based on a temporal logic formula is to give the agent a reward of 1 for satisfying the entire formula, and a reward of 0 otherwise. However, this is an example of a sparse reward (Sutton and Barto 2018). If few (or no) non-zero rewards are given to an agent in the learning process, it will be unable to update its policy frequently enough to make progress on the task in the timescale of a typical experiment (even though it theoretically may converge to an optimal policy on an infinite timescale).

Ng, Harada, et al. 1999 introduces a method to address this problem called *reward shaping*. The authors define  $F : S \times A \times S \rightarrow \mathbb{R}$  to be a *potential-based* shaping function on an MDP  $M = (S, A, P, \iota_{\text{init}}, R, \gamma)$

if there exists a real-valued function  $\Phi : S \rightarrow \mathbb{R}$  such that for all  $s, s' \in S$  and  $a \in A(s)$ , we have:

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s).$$

They then show that every optimal policy in  $M$  is also an optimal policy in  $M' = (S, A, P, \iota_{\text{init}}, F + R, \gamma)$  for any potential-based shaping function  $F$ , and vice versa. Reward shaping is a powerful tool to overcome sparse rewards in RL. It has also been used with some success in objective-based RL (Camacho, Icarte, et al. 2019).

The Simulation Lemma states that for any given MDP  $M$  with transition function  $T$ , and any  $\epsilon > 0$ , one can always find an MDP  $\hat{M}$  with transition function  $\hat{T} \neq T$  (and that is otherwise identical to  $M$ ), such that the optimal policy in  $\hat{M}$  results in a policy that is  $\epsilon$ -close to optimal in  $M$ . Using the Simulation Lemma, it is possible to put bounds on the number of sampled experiences required in RL for the learned policy to be an  $\epsilon$ -close approximation to an optimal policy (Abbeel and Ng 2005). The reason that this lemma does not hold in general for LTL interpreted on infinite traces, is that there are situations where the optimal policy depends on whether a particular environment transition probability is exactly 1, or less than 1. While this difference may be arbitrarily small, in the case of infinite traces, it can be the difference between a sum of rewards converging or not. Since the number of learning episodes required to distinguish 1 from less than 1 is unbounded, an  $\epsilon$ -close to optimal policy cannot be found in finite time (see Littman et al. 2017, Sec 1.2 for an example illustrating this phenomenon). This prevents one from making guarantees about learning an  $\epsilon$ -close to optimal policy in objective-based RL with an LTL objective in finite time. Nonetheless, it is still useful to develop objective-based RL algorithms that converge to an optimal policy in the limit. In the case of finite LTL, all reward summations are bounded, and so the Simulation Lemma holds.

### 4.3 Objective-based RL for LTL

In this section, we investigate methods proposed to handle the LTL satisfaction problem in RL. We find that most, but not all, methods leverage automata theory. We also explore how desirable properties in this setting such as *safe exploration* and continuous state and action spaces have been addressed in the literature.

### 4.3.1 Satisfaction Guarantees without Automata

The approach proposed in Littman et al. 2017 aims to find a solution to **G2** without using automata theory. To overcome the problem of the Simulation Lemma not holding for LTL on infinite traces, the authors augment the temporal operators of LTL, resulting in a new language they call Geometric Linear Temporal Logic (GLTL). Each temporal operator in GLTL is parameterised by a geometric distribution. This has the effect of transforming an objective “eventually  $b$  holds” to “ $b$  eventually holds within  $k$  timesteps”, where  $k$  is a random variable following a geometric distribution.

To solve **G2**, the authors first create a *specification MDP* with accepting and rejecting states which encodes the LTL formula. This process is illustrated recursively, by first considering how a single atomic proposition can be converted to a specification MDP, and then by defining rules for the treatment of logical and temporal operators. The resulting specification MDP is then combined with the MDP representing the environment, forming a *product MDP*. In the product MDP, states that correspond to an accepting or rejecting state of the specification MDP are converted to absorbing states (i.e. they transition to themselves with probability 1). A reward of +1 is given to states that correspond to an accepting state in the specification MDP. The authors show by construction how this reward scheme on the product MDP achieves **G2**. This approach also encourages the agent to satisfy the specification quickly, so as to avoid the expiration of temporal operators (and the subsequent task failure).

However, the speed with which an optimal policy completes the objective is entirely dependent on the settings of the temporal operator parameters, and there is no suggested approach for how to set such parameters to provide guarantees on the time taken for an optimal agent to satisfy an objective.

Another limitation of this article is the lack of any experimental validation of the proposed method. This makes it difficult to compare it to finite LTL satisfaction approaches in terms of time taken for an optimal agent to complete an objective. Furthermore, experiments that vary the parameter values of the temporal operators would illustrate the trade-off between satisfaction probability and objective completion time.

### 4.3.2 Automata-Based Approaches

Automata-based approaches to solving LTL satisfaction problems with model-free RL began with Sadigh et al. 2014. In this work, the authors consider vanilla LTL, and do not apply geometric timeouts to the temporal operators. Instead of manually crafting a process to create a specification MDP, this work uses a result originally derived in Safra 1988, which states that a *deterministic Rabin automaton* (DRA) can



be constructed from any LTL formula  $\varphi$  built from atomic propositions  $\mathcal{AP}$ , such that the DRA accepts exclusively all the infinite words over  $\mathcal{AP}$  that satisfy  $\varphi$ . The authors then create a product MDP by combining the DRA with the original MDP. The acceptance condition of the DRA is used to craft a reward function on the product MDP. Using this construction, the authors claim that if there exists a policy that satisfies an LTL formula with probability 1, then there also exists appropriate weightings for the reward function such that maximising the rewards on the product MDP will lead to a such a policy. The algorithm introduced in this work to find an optimal policy is based on temporal difference learning.

The authors validated their method experimentally on a gridworld environment, and an example traffic network. These experiments show that the approach allows the learning of optimal policies given non-trivial objectives expressed in LTL. The objectives used in the experiments are not of a finite nature, and so one cannot discuss the time taken to satisfy these objectives.

While this work is more general than that of Littman et al. 2017 in the sense that it operates on regular LTL rather than LTL with special temporal operators, the satisfaction goal is much weaker, aiming to achieve **G1** but not **G2**.

The satisfaction guarantee claimed in this paper was later shown to be invalid (Hahn et al. 2019, Section 3). In particular, it is shown that the presence of rejecting transitions in the Rabin condition can prevent the finding of an optimal policy. The authors suggest a remedy by using a *limit-deterministic Büchi automaton* (LDBA)<sup>1</sup> in place of a DRA. Terminal states are added to the LDBA, with a positive reward for transitions from an accepting state to the terminal state. The authors then show that by letting the transition probability from the accepting states to the terminal state approach zero, the agent is incentivised to visit an accepting state infinitely often to obtain the positive terminal reward, and, in turn, maximises the probability of visiting an accepting state (and thus satisfying the LTL formula corresponding to the LDBA).

A K Bozkurt et al. 2020 leverages the insights from Sadigh et al. 2014, as well as the theoretical improvements of using LDBAs in place of DRAs, to form a solution that satisfies **G2**. The authors create a reward and discounting scheme such that the agent’s value function in a particular state is equal to the probability of satisfying the Büchi condition in the LDBA as the reward discount factor approaches 1. A value function with this property breaks the reliance of the learning algorithm on tracking the transition probabilities, and allows the use of any model-free RL algorithm to solve the problem.

---

<sup>1</sup>In Hahn et al. 2019 the authors refer to *suitable limit-deterministic Büchi automata* (SLDBWs), but most of the literature simply use LDBAs.

### 4.3.3 Miscellaneous RL Settings for LTL

The problem of safe exploration in RL is motivated by the need for an agent to avoid potentially catastrophic actions during learning. This problem has recently gained much attention, as researchers prepare for the prospect of training RL agents in the real world and public online environments. An overview of safe exploration is given in Amodei et al. 2016, and several methods are surveyed in Garca and Fernández 2015.

Hasanbeig, Abate, et al. 2020a shows that safe exploration is also possible in the problem of satisfying **G2**. As in A K Bozkurt et al. 2020, the authors form a product MDP with the original MDP and an LDBA generated by the LTL formula. They then train an *optimistic learner*, which is a Q-learning agent as in A K Bozkurt et al. 2020, and a *pessimistic learner* which, at each timestep, calculates a set of permissible safe actions that the optimistic learner can take. The authors show that, even with this extra restriction on the learning agent, their algorithm ensures satisfaction of **G2**. This guarantee comes at a computational cost, however, as the agent must store and update estimated transition probabilities for a subset of the MDP. Despite this cost, the authors are able to validate the method in experiments on a slippery gridworld and the rich Pacman environment. The experiments show that not only does the agent better avoid unsafe states during learning, but this avoidance actually improves the time taken for the agent to learn a good policy in these environments. It is worth noting that these are *soft* safety constraints; there is no guarantee that the learned policy will avoid the unsafe states.

The authors in Alper Kamil Bozkurt et al. 2021 show that **G2** can be achieved even on a generalisation of an MDP which caters for adversarial inputs.

In Wang et al. 2020, the authors propose an approach that can handle LTL objectives in continuous state and action spaces. The propositional labels corresponding to environment states are given by pre-defined regions in the state space. For example, a proposition may be labelled as *true* whenever the agent is inside a two-dimensional square or circle. The algorithm introduced by the authors is shown to find policies that satisfy simple objectives in two-dimensional gridworld experiments with a car-like robot. However, the authors observed that their algorithm was not robust to certain configurations of the environment. Moreover, this work failed to offer any theoretical guarantees for the approach.

A domain which has drawn much attention recently is that of inverse reinforcement learning (Ng, Russell, et al. 2000) with LTL objectives. In this setting, the goal is to infer an LTL formula based on demonstrations of trajectories that both satisfy and do not satisfy the formula. Works such as Camacho

and McIlraith 2019; Shah et al. 2018; Chou et al. 2020 show that not only do viable solutions for this problem exist, but in the case of Chou et al. 2020, solutions are possible given only positive sub-optimal demonstrations.

## 4.4 Objective-based RL for Finite LTL

Given the focus in the literature on LTL, RL with finite LTL objectives has not been as widely studied.

In Giacomo, Iocchi, et al. 2019, the authors consider RL with  $LTL_f/LDL_f$  objectives in the *restraining bolt* setting; where the agent itself is unaware of the objective, and uses additional rewards given to it by the restraining bolt, which itself does not have a model of the environment nor the agent, but knows the objective and can view traces of the agent. The authors show theoretically and experimentally that model-free RL algorithms can be used to learn policies that satisfy  $LTL_f/LDL_f$  formulae, despite the agent not having direct access to the formula. It is unclear how the methods proposed handle the case where the optimal policy cannot satisfy the objective 100% of the time, as in **G2**.

### 4.4.1 Direct Translation to Rewards

In X. Li et al. 2017, the authors devise Truncated Linear Temporal Logic (TLTL), which is a variation of LTL that is interpreted over finite traces. As opposed to being defined on atomic propositions, TLTL is defined on *predicates* of the form  $f(s) < c$ , where  $f : S \rightarrow \mathbb{R}$  and  $c$  is a constant. Predicates allow the expression of real-valued metrics such as distance, energy, temperature, as opposed to the purely boolean semantics supported by languages such as LTL and  $LTL_f$ . TLTL is equipped with a quantitative semantics, also known as a *robustness degree*, which is a function that takes as input both a finite trace and a TLTL formula, and returns a real value. A positive output indicates that the finite trace satisfies the TLTL formula, whereas a negative output indicates that it does not, with the magnitude of the output giving a measure of how well the trace satisfies the formula.

The quantitative semantics is particularly well suited to domains with continuous state and action spaces. For example, given an objective such as “Eventually reach state A while avoiding state B”, the semantics will be negative for any traces which do not satisfy the goal. Larger negative magnitudes would be given to traces that are very far from satisfying the goal, such as traces representing the agent entering state B or never getting close to A. Even in simple examples like this which lack complex interactions between

temporal and propositional operators, the authors show that this method outperforms reward-based RL with rewards defined heuristically using Euclidean distances.

The strength of this approach is that it is model-free, can operate in continuous state and action spaces, and is not heavily limited by the size of the state space. This environmental flexibility comes at the cost of a lack of theoretical guarantees on performance. The article is only able to show experimentally that the approach can outperform heuristically designed reward functions, and it is thus unknown which domains this technique can be effectively applied to. Moreover, the lack of theory in the article raises the question of whether there are MDPs and predicates for which the proposed quantitative semantics will maximise the probability of satisfying a given TLTL formula. Despite operating on finite traces, this approach is incapable of offering assurances on whether the learned policy will complete the task as quickly as possible, or within some time bound.

## 4.5 Reward Machines

The authors of Camacho and McIlraith 2019; Icarte et al. 2020 show that reward machines (RMs) (Definition 15) can be leveraged to efficiently learn policies in objective-based RL problems. Camacho and McIlraith 2019 shows that any DFA (Definition 14) can be converted to an RM, meaning that RMs are compatible with objectives in any logic whose formulas can be expressed with DFAs, including all of the finite LTL formalisms.

These works create an RMxMDP (Definition 17) from the DFA corresponding to a temporal logic formula and the MDP, such that each state of the RMxMDP is a (environment\_state, RM\_state) tuple, and transitions in the RMxMDP depend on both the transitions of the MDP and the DFA. A Q-learning baseline is then run on the RMxMDP, which learns a policy that maximises rewards in the RM, and thus maximises the probability of satisfaction of the temporal logic formula.

The authors extend this Q-learning baseline to create the *counterfactual experience for reward machines* (CRM) algorithm, which we present a version of in Algorithm 4 in Section 5.2. The key improvement over the cross-product baseline is that, not only is an agent experience  $(s, a, s')$  used for updating the Q-value in the current RM state, but the same experience is used to update the Q-values for every state in the RM. This can be done as the MDP transitions are entirely independent of the RM transitions, and so knowledge about an MDP transition is useful to the agent regardless of the current state of the RM.

Note that, while each transition is independent of the RM state, a single experience may give rise to a different reward depending on the RM state.

As an added improvement to the CRM algorithm, the authors in Camacho and McIlraith 2019 equip a simple reward machine with extra rewards using reward shaping (Section 4.2). An MDP is constructed from the RM itself, with rewards of 1 given to transitions that are valid according to the RM, and 0 otherwise. The potential function is then set using the optimal state-values (that can be obtained by value iteration):  $\Phi(s, q) = -v^*(q)$ . Reward shaping encourages the agent to transition towards RM states that are closer to completing the objective.

Not only can these methods be applied to MDPs in tabular environments such as grid-worlds, but also on more complex environments such as those with continuous state spaces that require neural network approximations for policies. Minor adjustments are made to the deep RL algorithms Double DQN (Hasselt et al. 2016) and DDPG (Lillicrap et al. 2015) to handle counterfactual experiences and reward shaping, and the resulting algorithms are shown in Icarte et al. 2020 to improve on the Q-learning baseline in two continuous domains.

Another example showing the flexibility of reward machines is given in Icarte et al. 2018a. The authors tackle the problem of multi-task RL (Wilson et al. 2007), and show how converting objectives in co-safe LTL to reward machines can aid in transferring policies and experiences between different tasks. In this setting, CRM uses a single  $(s, a, s')$  experience to update the Q-values for each RM state in all RMs, not just the RM corresponding to the current task.

While not stated explicitly in the RM literature, when one applies the proposed methods to an objective given by a finite LTL formula, one can learn a policy that maximises the probability of satisfying the formula in an infinite horizon. The finite horizon problem (to maximise the satisfaction probability before a deadline) is not considered in this work. Fortunately, extending this work to the finite horizon problem can be done without sacrificing the ability to use counterfactual experiences and reward shaping. The RM literature also does not analyse the empirical value of counterfactual experiences compared to real experiences, an investigation of which may help refine the learning algorithms. In the following sections, we devise methods using RMs to handle the finite horizon problem, and evaluate the sample efficiency and update efficiency (indicating the relative value of the counterfactual experiences) of infinite horizon and finite horizon solutions in slippery gridworld experiments.

## 4.6 Planning with Finite LTL

In the setting where the agent has direct access to the transition function, several approaches have been proposed for satisfying finite LTL formula. In Wells et al. 2020, the authors outline two approaches. In one, the  $LTL_f$  formula is converted to an LTL formula using the translation outlined in Giacomo and Vardi 2013, and standard techniques are used to solve the LTL planning problem, such as those described in C. Baier and Katoen 2008. In the other, the  $LTL_f$  formula is directly converted to a DFA, a product MDP is created with the DFA and the original MDP, and a policy is found (often called a plan or a controller in this setting) that maximises the probability of reaching an accepting state of the DFA. The authors evaluate these methods experimentally and find that the latter approach offers better scalability, despite both having the same theoretical complexity bounds. Although not stated by the authors, since the latter approach does not rely on a conversion to LTL, it can be extended to handle time-related constraints or goals using methods described in Chapter 5.

The authors in Lacerda et al. 2015 develop an approach that concurrently maximises the probability of satisfaction of a co-safe LTL formula, and minimises the cost of achieving the task. Trade-offs between the objectives are handled by the use of *constrained queries*, which, for example, sets a threshold of minimal satisfaction probability and tries to minimise the cost given that threshold. Moreover, the authors also give consideration to a *progression metric*, which measures how much of a task is completed even if it is not fully satisfied. The approach first forms a product MDP between the DFA generated from the co-safe LTL formula, and the original MDP. This MDP is then *trimmed* by removing states in which it is no longer possible to achieve a reward. The multi-objective problem on the trimmed MDP is then solved using the symbolic model checking tool PRISM (Kwiatkowska et al. 2011). An extension of these properties to the RL setting would be highly valuable to the community, but would not come without challenges; model checking tools rely on known transition dynamics, and the corresponding multi-objective RL algorithms may be slow to train on the large state spaces created by the product MDP.

## 4.7 RL with LTL Landscape

In Table 4.1, we present a summary of several works that attempt to satisfy LTL and finite LTL objectives with RL. The table excludes works that attempt to satisfy LTL objectives when the environment

transitions are known (planning), as well as settings such as IRL which do not aim to satisfy a given formula.

Article	Language	Trace Type	Satisfaction Guarantee	State Space
Hasanbeig, Abate, et al. 2020a	LTL	Infinite	<b>G2</b>	Finite
Hasanbeig, Kantaros, et al. 2019	LTL	Infinite	<b>G2</b>	Finite
Hasanbeig, Abate, et al. 2018	LTL	Infinite	<b>G1</b>	Finite
Wang et al. 2020	LTL	Infinite	<b>G1</b>	Continuous
Oura et al. 2020	LTL	Infinite	<b>G1</b>	Finite
A K Bozkurt et al. 2020	LTL	Infinite	<b>G2</b>	Finite
Sadigh et al. 2014	LTL	Infinite	N/A	Finite
Hasanbeig, Abate, et al. 2020b	LTL	Infinite	<b>G2</b>	Continuous
Alper Kamil Bozkurt et al. 2021	LTL	Infinite	<b>G2</b>	Finite
Hahn et al. 2019	LTL	Infinite	<b>G2</b>	Finite
Littman et al. 2017	GLTL	Infinite	<b>G2</b>	Finite
Icarte et al. 2018a	co-safe LTL	Finite	N/A	Finite
Camacho and McIlraith 2019	finite LTL	Finite	<b>G2</b>	Both
Icarte et al. 2020	finite LTL	Finite	<b>G2</b>	Both
X. Li et al. 2017	TLTL	Finite	N/A	Continuous

TABLE 4.1. Summary of works in objective-based RL addressing the infinite horizon problem. The trace type column indicates whether a finite or infinite trace is needed to verify satisfaction of the formula.

## 4.8 Finite Horizon Problem in Reward-Based RL

While the objective-based RL literature has appeared silent on handling problems with fixed deadlines, there has been some work in the reward-based RL literature on this problem.

In Harada 1997, the author first introduces a baseline for solving the reward-based finite horizon problem. For a deadline  $H$ , the baseline method runs Q-learning on an MDP with a state space obtained by taking the product of the MDP state space with a set of timesteps  $h \in \{0, 1, \dots, H\}$ . The Q-value  $q_h(s)$  is then updated for each experience  $(h, s, a, s')$ . The author then introduces the QT-learning algorithm, which uses a similar trick to CRM (see Section 4.5 and Algorithm 4) to improve sample efficiency; instead of only making a single update using the experience  $(h, s, a, s')$ , updates are made for all Q-values  $q_{h'}(s, a)$  with  $h' \in \{0, 1, \dots, H - 1\}$ . This can be done due to the fact that the environment transitions in an MDP depend only on the action and current state, and are thus independent of the current timestep.

### 4.8.1 Time-Awareness in Deep RL

Despite this early observation in Harada 1997, the reinforcement learning community has generally failed to account for time limits in their implementations. While ignoring time limits reduces the computational overhead, it comes at the sometimes heavy cost of converging to policies that are far from optimal in a finite horizon setting. The authors in Pardo et al. 2018 reminded the RL community of this fact, and showed how costly ignoring time constraints can be in modern deep RL environments. They show that by simply giving the agent access to the number of timesteps remaining in an episode (making the policy time-dependent), the time-aware agents are able to outperform non-time-aware agents on several of OpenAI’s MuJoCo Gym environments (Brockman et al. 2016). They present video samples showing that a time-aware Hooper-v1 agent – a one legged agent tasked to maximise its forward distance by hopping – is able to lean forward just before the finish line, as is done by professional sprinters.

While these works address the finite horizon problem for reward-based RL problem, where a numerical reward function can be specified by the designer, their methods provide a strong base for which to extend to finite horizon objective-based RL problems. We extend these works to handle such problems in the following sections.



## Methods

---

In this section, we leverage the reward machine framework to provide solutions to the finite and infinite horizon problems in objective-based RL. In the sections that follow, we:

- (1) Show that the infinite horizon problem can be solved directly using reward machines along with any RL algorithm that guarantees optimality, such as Q-learning or CRM (Section 5.1).
- (2) Present a solution to the finite horizon problem by making use of reward machines as well as results from reward-based RL (Section 5.1).
- (3) Introduce the QTRM-learning algorithm for the finite horizon problem which exploits the independence between environment transitions, reward machine states, and the timestep counter (Algorithm 5).
- (4) Examine the implications of the time dependence of optimal policies in the finite horizon problem. In particular, we show that modifying a memoryless infinite horizon solution using heuristic methods, such as reward discounting or step-based penalties, cannot guarantee the learning of optimal policies in a finite horizon problem (Section 5.3).

### 5.1 Objective-Based RL Theorems

In order to solve the infinite horizon objective problem, we make use of the following lemma from C. Baier and Katoen 2008, Sec 10.6.1.

**LEMMA 1 (Reachability).** *Consider the MDP  $M = (S, A, P, t_{init})$  and a finite set of goal states  $B \subseteq S$  with  $P(s, a, s) = 1$  for all  $s \in B, a \in A(s)$ . Suppose  $(x_s)_{s \in S}$  is the vector such that  $x_s = \Pr^{max}(s \models \diamond B)$*

is the maximum probability of reaching a state in  $B$  when starting from state  $s$ .<sup>1</sup> If  $s \not\Leftarrow \Diamond B$  denotes that  $B$  is not reachable from  $s$ , then the following hold for  $s \in S$ :

- (1) If  $s \in B$ , then  $x_s = 1$
- (2) If  $s \not\Leftarrow \Diamond B$ , then  $x_s = 0$
- (3) Otherwise,  $x_s = \max_a \sum_{s' \in S} P(s, a, s') \cdot x_{s'}$ .

PROOF. The first two conditions hold by the axioms of probability. The third condition follows by the fact that environment transitions depend only on the current state and action. This property allows for the reachability probability of one state to be calculated using the transition probabilities from all adjacent states and their respective reachability probabilities. By mathematical induction, one can show that the condition holds by bootstrapping from the values of  $x_s$  given by the first two conditions (C. Baier and Katoen 2008, Sec 10.6.1).  $\square$

This lemma shows that to get the maximum probabilities, one must solve a system of equations of length  $|S|$ . Also of note is that  $x_s$  is the unique solution to this system of equations (C. Baier and Katoen 2008, Sec 10.6.1).

**THEOREM 5.1.1 (Infinite Horizon Objective Problem).** *Consider the MDP  $M = (S, A, P, \iota_{\text{init}}, \gamma, \mathcal{AP}, L)$  with  $\gamma = 1$ , and an objective expressed in a finite LTL formula  $\varphi$ .<sup>2</sup> Suppose  $\mathcal{R}_M = (U, u_0, B, \delta_u, \delta_r)$  is the simple reward machine corresponding to  $\varphi$  and  $M$ , with  $\delta_r(u, L(s, a, s')) = 1$  if  $\delta_u(u, L(s, a, s')) \in B$  and  $u \notin B$ , and 0 otherwise (i.e. a reward of 1 for the transition that first satisfies the objective, and 0 otherwise). Let  $\mathcal{T} = (S, A, P, \iota_{\text{init}}, \gamma, \mathcal{AP}, L, U, u_0, B, \delta_u, \delta_r)$  be the MDPRM obtained from  $M$  and  $\mathcal{R}_M$ . Then the optimal policy in  $\mathcal{T}$  also maximises the probability of satisfying  $\varphi$ .*

PROOF. We form the RMxMDP  $M_{\mathcal{T}} = (S', A', P', \iota'_{\text{init}}, R', \gamma')$  from  $\mathcal{T}$  as in Definition 17. We also introduce a set of goal states  $B' = \{(s', u') : \delta_u(u, L(s, a, s')) = u' \in B\} \subseteq S'$  for some  $a \in A'(s), s \in S$  and  $u \in U$ . First, we observe that any policy in  $M_{\mathcal{T}}$  achieves the same rewards as a policy in  $\mathcal{T}$ . Therefore, an optimal policy in  $M_{\mathcal{T}}$  is also an optimal policy in  $\mathcal{T}$ . Second, we note that

<sup>1</sup>This notation can be interpreted in two ways. As in C. Baier and Katoen 2008, Section 10.6 page 845, it can refer to a probability space defined on an MDP which has a single initial state  $s$  (i.e.  $\iota_{\text{init}}(s) = 1$ ). Alternatively, it can refer to the probability space defined on an MDP with any initial state distribution (provided state  $s$  is reachable from an initial state), with  $\Pr(s \Leftarrow \Diamond B)$  representing the probability of reaching a state in  $B$  conditioned on first visiting  $s$ . Since MDP transitions are independent of prior transitions, these two formulations are equivalent.

<sup>2</sup>Recall that a finite LTL formula is a formula in co-safe LTL, LTL<sub>f</sub>, or LDL<sub>f</sub>.

satisfying  $\varphi$  is equivalent to reaching a state in  $B'$ . Therefore, we need to show that an optimal policy in  $M_{\mathcal{T}}$  also maximises the probability of reaching a state in  $B'$ .

Let  $(x_{(s,u)})_{(s,u) \in S'}$  be a vector such that  $x_{(s,u)} = \Pr^{\max}((s,u) \models \diamond B')$  is the maximum probability of reaching a state in  $B'$  from the state  $(s,u) \in S'$ .

Let  $\pi$  be an optimal policy in  $\mathcal{T}$ . Since  $\pi$  is optimal, it satisfies the Bellman optimality equations. In particular, we have that for all  $(s,u) \in S'$ ,

$$V^{\pi}((s,u)) = \max_a \sum_{(s',u') \in S'} P'((s',u')|(s,u),a)[R'((s,u),a,(s',u')) + V^{\pi}((s',u'))]. \quad (5.1)$$

Note that  $R'((s,u),a,(s',u')) = 1$  precisely when  $(s,u) \notin B'$  and  $(s',u') \in B'$ , and 0 otherwise. Using the results from Lemma 1 we show the equivalency between the optimal value function  $V^{\pi}$  and the vector  $(x_{(s,u)})_{(s,u) \in S'}$ . It is enough to show that  $V^{\pi}((s,u)) = x_{(s,u)}$  for all states  $(s,u) \in S' \setminus B'$ , as the terminal states  $B'$  can only self-loop and thus cannot change the policy.

If  $(s,u) \notin \diamond B'$ , the MDP can never reach  $B'$  to obtain any rewards, and thus  $V^{\pi}((s,u)) = 0 = x_{(s,u)}$ . If  $(s,u) \in S' \setminus B'$  and  $(s,u) \models \diamond B'$ , then from Equation 5.1 we have  $R'((s,u),a,(s',u')) + V^{\pi}((s',u')) = 1 + 0 = 1$  if  $(s',u') \in B'$ , and  $V^{\pi}((s',u'))$  otherwise. Therefore, by Lemma 1, we have that  $V^{\pi}((s,u)) = x_{(s,u)}$  for all  $(s,u) \in S' \setminus B'$ , and thus  $\pi$  also maximises the probability of reaching a state in  $B'$ .  $\square$

The converse of Theorem 5.1.1 is also true; a policy that maximises the probability of satisfying  $\varphi$  is also an optimal policy in  $\mathcal{T}$ . This can be shown in much the same way, but instead, starting with a policy that satisfies the system of equations in Lemma 1 and showing that it also satisfies the Bellman optimality equations.

We can extend the above theorem to the RL setting as follows:

**COROLLARY 5.1.1.1.** *Consider the same setup as in Theorem 5.1.1. Then any RL algorithm that converges to an optimal policy in  $\mathcal{T}$ , such as Q-learning and CRM, also maximises the probability of satisfying  $\varphi$ .*

This corollary shows that one need only sample from the transition function  $P$  in the MDP in order to learn a policy that maximises the satisfaction probability of a finite LTL objective  $\varphi$ .

Another important observation is that the optimal policy is memoryless in  $M_{\mathcal{T}}$ , which has the state space  $S' = S \times (U \cup B)$ . As we will see in Section 5.3, this prevents the use of common heuristic methods to speed up the agent and achieve optimality in the finite horizon problem, which can require a time-dependent policy.

Before introducing the equivalent theorem for finite horizon problems, we first present the following reachability lemma for the step-bounded case.

**LEMMA 2 (Step-Bounded Reachability).** *Consider the MDP  $M = (S, A, P, \iota_{init})$ , with a finite horizon  $H$ . Let  $B \subseteq S$  be a set of goal states with  $P(s, a, s) = 1$  for all  $s \in B$  and  $a \in A(s)$ . Let  $(x_s^h)_{s \in S, h \in \{0, 1, \dots, H\}}$  be the vector such that  $x_s^h = \text{Pr}^{\max}(s \models^{\leq H-h} \diamond B)$  represents the maximum probability of reaching a state in  $B$  from  $s \in S$  within  $H - h$  timesteps. Then the following hold for  $s \in S$ :*

- (1) *If  $s \in B$ , then  $x_s^h = 1$  for all  $h \in \{0, 1, \dots, H\}$*
- (2) *If  $h = H$  and  $s \in S \setminus B$ , then  $x_s^h = 0$*
- (3) *If  $h < H$  and  $s \in S \setminus B$ , then  $x_s^h = \max_a \sum_{s' \in S} P(s, a, s') \cdot x_{s'}^{h+1}$ .*

**PROOF.** This lemma holds for the same reasons as Lemma 1, with the additional (obvious) fact that environment transitions take exactly one timestep.  $\square$

We now state the solution to the finite horizon problem in objective-based RL, the proof of which follows the same structure as the infinite horizon case.

**THEOREM 5.1.2 (Finite Horizon Objective Problem).** *Consider the MDP  $M = (S, A, P, \iota_{init}, \gamma, \mathcal{AP}, L)$  with  $\gamma = 1$ . Let  $H$  be the finite horizon of  $M$ , and  $\varphi$  be a finite LTL formula expressing the agent's objective. Suppose  $\mathcal{R}_M = (U, u_0, B, \delta_u, \delta_r)$  is the simple reward machine corresponding to  $\varphi$  and  $M$ , with  $\delta_r(u, L(s, a, s')) = 1$  if  $\delta_u(u, L(s, a, s')) \in B$  and  $u \notin B$ , and 0 otherwise. Let  $\mathcal{T} = (S, A, P, \iota_{init}, \gamma, \mathcal{AP}, L, U, u_0, B, \delta_u, \delta_r)$  be the finite horizon MDPRM obtained from  $M$  and  $\mathcal{R}_M$ . Then the optimal policy in  $\mathcal{T}$  also maximises the probability of satisfying  $\varphi$  within  $H$  timesteps.*

**PROOF.** We form the finite horizon RMxMDP  $M_{\mathcal{T}} = (S', A', P', \iota'_{init}, R', \gamma')$  from  $\mathcal{T}$  as in Definition 17. We also introduce a set of goal states  $B' = \{(s', u') : \delta_u(u, L(s, a, s')) = u' \in B \text{ for some } a \in A'(s), s \in S \text{ and } u \in U\}$ . As in the infinite horizon case, we observe that any policy in  $M_{\mathcal{T}}$  achieves the same rewards as a policy in  $\mathcal{T}$ , and thus, an optimal policy in  $M_{\mathcal{T}}$  is also an optimal policy in  $\mathcal{T}$ . We also note that satisfying  $\varphi$  within  $H$  timesteps is equivalent to reaching a state in  $B'$  within  $H$

timesteps. Therefore, we need to show that an optimal policy in  $M_{\mathcal{T}}$  also maximises the probability of reaching a state in  $B'$  within  $H$  timesteps.

Let  $(x_{(s,u)}^h)_{(s,u) \in S', h \in \{0,1,\dots,H\}}$  be a vector such that  $x_{(s,u)}^h = \Pr^{\max}((s,u) \models^{\leq H-h} \diamond B')$  is the maximum probability of reaching a state in  $B'$  from a state  $(s,u) \in S'$  within  $H - h$  timesteps.

Let  $\pi$  be an optimal policy in  $\mathcal{T}$ . Since  $\pi$  is optimal, it satisfies the Bellman optimality equations. In particular, we have that, for all  $(s,u) \in S'$ ,  $V_H^\pi((s,u)) = 0$  and for  $h \in \{0,1,\dots,H-1\}$ ,

$$V_h^\pi((s,u)) = \max_a \sum_{(s',u') \in S'} P'((s',u')|(s,u),a) [R'((s,u),a,(s',u')) + V_{h+1}^\pi((s',u'))]. \quad (5.2)$$

Note that  $R'((s,u),a,(s',u')) = 1$  precisely when  $(s,u) \notin B'$  and  $(s',u') \in B'$ , and 0 otherwise. Using the results from Lemma 2 we show that  $(x_{(s,u)}^h) = V_h^\pi((s,u))$  for all states  $(s,u) \in S' \setminus B'$  and  $h \in \{0,1,\dots,H\}$  (again, the terminal states  $B'$  can only self-loop and thus cannot change the policy).

If  $h = H$  and  $(s,u) \in S' \setminus B'$ , we have  $V_H^\pi((s,u)) = 0 = x_{(s,u)}^H$ . If  $h < H$  and  $(s,u) \in S' \setminus B'$ , then, from Equation 5.2, we have  $R'((s,u),a,(s',u')) + V_{h+1}^\pi((s',u')) = 1 + 0 = 1$  if  $(s',u') \in B'$ , and  $V_{h+1}^\pi((s',u'))$  otherwise. Therefore, by Lemma 2 we have that  $V_h^\pi((s,u)) = x_{(s,u)}^h$  for all  $(s,u) \in S' \setminus B'$  and  $h \in \{0,1,\dots,H\}$ , and thus  $\pi$  also maximises the probability of reaching a state in  $B$  within  $H$  timesteps.  $\square$

As in the infinite horizon case, one can easily prove the converse for Theorem 5.1.1; a policy that maximises the probability of satisfying  $\varphi$  within  $H$  timesteps is also an optimal policy in  $\mathcal{T}$ . Also, we can extend this theorem to the RL setting in the same way:

**COROLLARY 5.1.2.1.** *Consider the same setup as in Theorem 5.1.2. Then any RL algorithm that converges to an optimal policy in  $\mathcal{T}$ , also maximises the probability of satisfying  $\varphi$  within  $H$  timesteps.*

## 5.2 Objective-Based RL Algorithms

In this section, we introduce some of the algorithms that can be used to solve the objective-based RL problems. As noted in Corollaries 5.1.1.1 and 5.1.2.1, any algorithm that converges to an optimal policy in the RMxMDPs  $\mathcal{T}$  is sufficient.

For the infinite horizon problem, we present the CRM algorithm (Algorithm 4) (Icarte et al. 2020).

**Algorithm 4** Infinite horizon Q-learning and CRM

---

**Input:** MDP  $\mathcal{T} = (S, A, P, \iota_{\text{init}}, \gamma, \mathcal{AP}, L, U, u_0, B, \delta_u, \delta_r)$  with  $\gamma \in [0, 1]$ , learning rate  $\alpha \in (0, 1]$ , small  $\epsilon > 0$

- 1: For all  $s \in S, a \in A, u \in U$ , initialise  $Q(s, u, a)$  arbitrarily
- 2: **for** each episode until termination condition **do**
- 3:     Sample starting environment state  $s$  from  $\iota_{\text{init}}$
- 4:     Initialise reward machine state  $u \leftarrow u_0$
- 5:     **while**  $u \notin B$  **do**
- 6:         Take action  $a$  using  $\epsilon$ -greedy policy on  $Q$  in the current state  $(s, u)$ , and observe  $s'$
- 7:         Compute the next RM state  $u' \leftarrow \delta_u(u, L(s, a, s'))$
- 8:         **if** CRM **then**
- 9:             experience  $\leftarrow \{(s, \bar{u}, a, \delta_r(\bar{u})(s, a, s'), s', \delta_u(\bar{u}, L(s, a, s')))\} | \forall \bar{u} \in U\}$
- 10:         **else if** Q-learning **then**
- 11:             experience  $\leftarrow \{(s, u, a, \delta_r(u)(s, a, s'), s', u')\}$
- 12:         **end if**
- 13:         **for**  $\langle s, \bar{u}, a, \bar{r}, s', \bar{u}' \rangle \in \text{experience}$  **do**
- 14:             **if**  $s'$  is terminal or  $\bar{u}' \in B$  **then**
- 15:                  $Q(s, \bar{u}, a) \leftarrow Q(s, \bar{u}, a) + \alpha[\bar{r} - Q(s, \bar{u}, a)]$
- 16:             **else**
- 17:                  $Q(s, \bar{u}, a) \leftarrow Q(s, \bar{u}, a) + \alpha[\bar{r} + \gamma \max_{a'} Q(s', \bar{u}', a') - Q(s, \bar{u}, a)]$
- 18:             **end if**
- 19:         **end for**
- 20:         Update  $s \leftarrow s'$  and  $u \leftarrow u'$
- 21:     **end while**
- 22: **end for**

**Output:** Optimal action values  $Q$

---

Algorithm 4 also contains the vanilla Q-learning algorithm applied to  $\mathcal{T}$  (lines 10-11). The key improvement of CRM over Q-learning are the counterfactual experiences used (lines 8-9). In CRM, not only is an agent experience  $\langle s, a, s' \rangle$  used for updating the Q-value in the current reward machine state, but the same transition is used to update the Q-values for every state in the reward machine (along with their RM state-specific rewards). This can be done as the MDP transitions are entirely independent of the RM transitions, and thus the counterfactual experiences are valid datapoints to update the policy and the convergence guarantee of Q-learning is maintained.

For the finite horizon problem, we introduce QTRM-learning (Algorithm 5). The QTRM-learning algorithm encompasses the sub-algorithms Q-learning, QT-learning, and QRM-learning. The difference between these algorithms is the type of counterfactual experiences used. In Q-learning, no counterfactual experiences are used (lines 7-8). Algorithms that include ‘T’ in the prefix use counterfactual experiences for each timestep (lines 11-12 and 13-14), and algorithms that include ‘RM’ in the prefix use counterfactual experiences for each reward machine state (lines 9-10 and 13-14). As in the infinite

**Algorithm 5** Finite horizon Q-learning, QRM-learning, QT-learning, QTRM-learning

---

**Input:** MDP  $\mathcal{T} = (S, A, P, \iota_{\text{init}}, \gamma, \mathcal{AP}, L, U, u_0, B, \delta_u, \delta_r)$  with  $\gamma \in [0, 1]$ , learning rate  $\alpha \in (0, 1]$ , small  $\epsilon > 0$ , horizon  $H$ , episodes\_per\_reset  $N$

- 1: For all  $s \in S, a \in A, u \in U$ , initialise  $Q_h(s, u, a)$  arbitrarily for all  $h \in \{0, 1, \dots, H - 1\}$
- 2: **for** each episode  $e$  until termination condition **do**
- 3:     Every  $N$  episodes (including the first), sample  $s \sim \iota_{\text{init}}$  and set RM state  $u \leftarrow u_0$
- 4:     **for** each step in episode  $h \in \{0, 1, \dots, H - 1\}$  **do**
- 5:         Take action  $a$  using  $\epsilon$ -greedy policy on  $Q_h$  in the current state  $(s, u)$ , and observe  $s'$
- 6:         Compute the next RM state  $u' \leftarrow \delta_u(u, L(s, a, s'))$
- 7:         **if** Q-learning **then**
- 8:             experience  $\leftarrow \{\langle h, s, u, a, \delta_r(u)(s, a, s'), s', u' \rangle\}$
- 9:         **else if** QRM-learning **then**
- 10:             experience  $\leftarrow \{\langle h, s, \bar{u}, a, \delta_r(\bar{u})(s, a, s'), s', \delta_u(\bar{u}, L(s, a, s')) \rangle \mid \forall \bar{u} \in U\}$
- 11:         **else if** QT-learning **then**
- 12:             experience  $\leftarrow \{\langle \bar{h}, s, u, a, \delta_r(u)(s, a, s'), s', u' \rangle \mid \forall \bar{h} \in [0, H - 1]\}$
- 13:         **else if** QTRM-learning **then**
- 14:             experience  $\leftarrow \{\langle \bar{h}, s, \bar{u}, a, \delta_r(\bar{u})(s, a, s'), s', \delta_u(\bar{u}, L(s, a, s')) \rangle \mid \forall \bar{u} \in U, \bar{h} \in [0, H - 1]\}$
- 15:         **end if**
- 16:         **for**  $\langle \bar{h}, s, \bar{u}, a, \bar{r}, s', \bar{u}' \rangle \in \text{experience}$  **do**
- 17:             **if**  $s'$  is terminal or  $\bar{u}' \in B$  or  $\bar{h} = H - 1$  **then**
- 18:                  $Q_{\bar{h}}(s, \bar{u}, a) \leftarrow Q_{\bar{h}}(s, \bar{u}, a) + \alpha[\bar{r} - Q_{\bar{h}}(s, \bar{u}, a)]$
- 19:             **else**
- 20:                  $Q_{\bar{h}}(s, \bar{u}, a) \leftarrow Q_{\bar{h}}(s, \bar{u}, a) + \alpha[\bar{r} + \gamma \max_{a'} Q_{\bar{h}+1}(s', \bar{u}', a') - Q_{\bar{h}}(s, \bar{u}, a)]$
- 21:             **end if**
- 22:         **end for**
- 23:         Update  $s \leftarrow s'$  and  $u \leftarrow u'$
- 24:     **end for**
- 25: **end for**

**Output:** Optimal action values  $Q$

---

horizon case, convergence guarantees of Q-learning hold in Q(TRM)-learning as the counterfactual experiences are valid datapoints due to the satisfaction of the Markov property for environment transitions. Note that we introduce the episodes\_per\_reset parameter  $N$  to increase the likelihood of the agent obtaining a reward and updating Q-values with non-zero values. This is especially useful early in training, where a random policy is very unlikely to complete the task before the horizon. A similar effect can be achieved by sampling the initial environment and RM state from a range of states during training, causing some episodes to begin with the agent already having progressed in the task.

The sample efficiency and number of updates required to learn good policies in all of the aforementioned algorithms are presented in Chapter 6.

In the next section, we show that while setting  $\lambda < 1$  is a useful heuristic method for the agent to prioritise near-term rewards, it violates Theorem 5.1.2.

### 5.3 Speeding Up the Agent with Heuristics

In the infinite horizon RL literature, it is common to reduce the discount factor  $\lambda$  or add negative rewards to every environment transition in order to incentivise the agent to collect rewards more quickly. In order to see why we cannot solve the finite horizon problem with these methods, we observe that an optimal policy in a finite horizon problem, as can be obtained by value iteration (Algorithm 1), can be time-dependent.

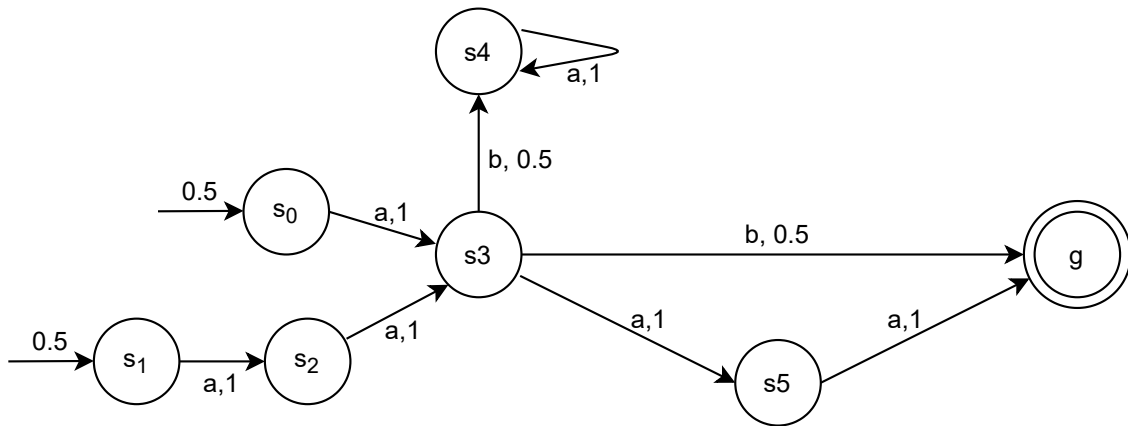


FIGURE 5.1. MDP with goal state  $g$  and initial state sampled uniformly from  $\{s_0, s_1\}$ . A reward of 1 is given for transitions to  $g$  and 0 otherwise. Edges are labelled with actions and their corresponding transition probabilities.

For the MDP illustrated in Figure 5.1, consider the optimal policy for the finite horizon problem with a horizon of 3, i.e. the policy that maximises rewards (and thus the reachability objective) within 3 steps. If the starting state is  $s_0$ , then the optimal policy at state  $s_3$  must take action  $a$  as it guarantees a reward after the future (assured) transitions to  $s_5$  and then  $g$ . However, if the starting state is  $s_1$ , then the optimal policy at  $s_3$  must take action  $b$  as two steps have already passed and the only chance of a reward is to get there in one step, even if it risks transitioning to a state that perpetually self-loops. This is an example of an optimal policy that is time-dependent. In contrast, a memoryless policy can only take into account the current state, and thus cannot determine whether to take action  $a$  or  $b$  in  $s_3$ . Therefore, no memoryless policies are optimal in this MDP.

In particular, regardless of the discount rate or step-based penalties, a memoryless optimal policy in the infinite horizon case is not necessarily an optimal policy in the finite horizon case. This is not to say that reward discounting or adding step-based penalties cannot usefully alter the learned policies. For example, the optimal policy for the infinite horizon problem with  $\gamma = 1$  will cause the agent to take



action  $a$  in  $s_3$ , whereas the optimal policy with  $\gamma < 0.5$  will cause the agent to take action  $b$  in  $s_3$ . This could be a preferable alteration if the designer wishes the agent to complete the task more quickly (at the risk of not completing it some of the time). Similarly, adding a reward of  $-\frac{2}{3}$  or smaller to each step will cause the agent to take action  $b$  in state  $s_3$ , as opposed to action  $a$  without step-based rewards. Section 6.4 analyses the effects of these methods empirically. Note that while we do not have a reward machine tied to this example, one can consider each state in the MDP as representing a state in an RMxMDP.

This result shows that a finite horizon optimal policy cannot, in general, be learned using these techniques in the infinite horizon setting. However, due to the additional computational cost of learning time-dependent finite horizon optimal policies compared to memoryless infinite horizon optimal policies, these heuristic methods applied to the infinite horizon setting may often be more practical than finite horizon solutions in RL applications.

## Experimental Evaluation

---

In this chapter, we empirically evaluate several methods to solve the objective-based infinite and finite horizon problems. In particular:

- (1) We introduce the Frozen Lake environment, a stochastic environment suitable for expressing finite LTL objectives that also allows easy interpretability of policies and state values. Using this environment, we specify objectives of varying complexity in finite LTL formulae (Section 6.1).
- (2) We convert the objectives from (1) to reward machines (Section 6.2).
- (3) We find optimal value functions and policies for these tasks using techniques from dynamic programming (Section 6.3).
- (4) We show the empirical utility of heuristic methods, such as reward discounting and step-based penalties, and explore issues that can arise when applying these methods (Section 6.4).
- (5) We compare the efficiency of various Q-learning based algorithms in terms of the number of samples and the number of Q-value updates required to learn good policies (Section 6.5).

### 6.1 Frozen Lake Environment

To evaluate the methods explored in this work, we developed a variation of the Frozen Lake environment (Brockman et al. 2016):

*You and your friends were tossing around a frisbee on a (mostly) frozen lake when your friend fell through the ice trying to take an epic catch. Luckily, somebody left rope on the lake for this exact scenario. You must grab a rope, and go over and pull your friend out of the water. The ice is slippery, so you will not always move in the*

*direction you intend. There are also some areas where the ice is thinner. If you step into one of those traps, you will fall in and drown.*

The parameters of the environment are described as follows:

- Actions: The agent can request to move one step in any of the 8 directions (Right, Down-Right, Down, ...).
- Light Blue Squares: Slippery ice, with 50% chance of an action leading to the desired state, and 50% chance of transitioning to a state uniformly from all 8 directions.
- Green Squares: Safe path, with all actions deterministic and leading to the desired state.
- Dark Blue Squares: Trap, where the agent would fall in and get stuck perpetually.
- Edge Conditions: If the direction of movement is off the grid, the agent will remain in place.

These parameters define the structure of the MDP  $M = (S, A, P, \iota_{\text{init}}, \mathcal{AP}, L)$  for the Frozen Lake environment, with  $S$  representing all the cells of the grid (25 in this example),  $A$  representing each of the 8 actions,  $P$  is deterministic when on the green path and the dark blue traps, and stochastic on the ice,  $\iota_{\text{init}}(s) = 1$  if  $s$  is the state in the top left corner, and 0 otherwise,  $\mathcal{AP} = \{F, R, T\}$  where  $F$  denotes the Friend atom,  $R$  denotes the Ropes, and  $T$  denotes the dark blue Traps, and  $L(s, a, s') = p$  for all  $s \in S$  and  $a \in A$  with  $P(s, a, s') > 0$  if  $s'$  is in a grid cell with atom  $p \in \mathcal{AP}$  (i.e. an atom is true when the agent visits the cell containing it).

We introduce three tasks using our Frozen Lake environment, shown in Figures 6.1, 6.2 and 6.3. We analyse properties of these tasks and run experiments in both the infinite and finite horizon settings. Note that the objective of each task can be expressed with a formula in any of the finite LTL formalisms.



FIGURE 6.1. Frozen Lake environment Task 1:  $\varphi_1 = \diamond F$  (i.e. eventually get to Friend)



FIGURE 6.2. Frozen Lake environment Task 2:  $\varphi_2 = \diamond(R \wedge \diamond F)$  (i.e. eventually get the Rope and then eventually get to Friend).



FIGURE 6.3. Frozen Lake environment Task 3:  $\varphi_3 = \diamond(R_1 \wedge \diamond(R_2 \wedge \diamond(R_3 \wedge \diamond F)))$  (i.e. eventually get the Ropes in order and then eventually get to Friend).

There is an alternative formulation of these objectives which expresses the fact that the agent should not enter a dark blue square (i.e. a trap). The corresponding formulae are simply a conjunction of  $\Box \neg T$  (meaning “always not T”) with the original formula. For example,  $\varphi_1 = \diamond F \wedge \Box \neg T$  and  $\varphi_2 = \diamond(R \wedge \diamond F) \wedge \Box \neg T$ . The reason we choose not to use this formulation, is that the environment will never allow the agent to leave a dark blue square, and thus the objective would be violated even without expressing this in the formula explicitly. Section 7 discusses the trade-offs for this decision.

Since Task 3 gives rise to rather complex policies, we use it only to compare the sample efficiency and computational efficiency of the algorithms in Section 6.5, and focus on Tasks 1 and 2 for the more exploratory finer-grained analysis in the next two sections.

## 6.2 Reward Machines for Tasks

In order to learn policies for our tasks, we must first convert the temporal logic objectives to reward machines following the process described in Section 2.4. The reward machine representation of the objectives of Task 1 and Task 2 are shown in Figure 6.4a and Figure 6.4b, respectively.

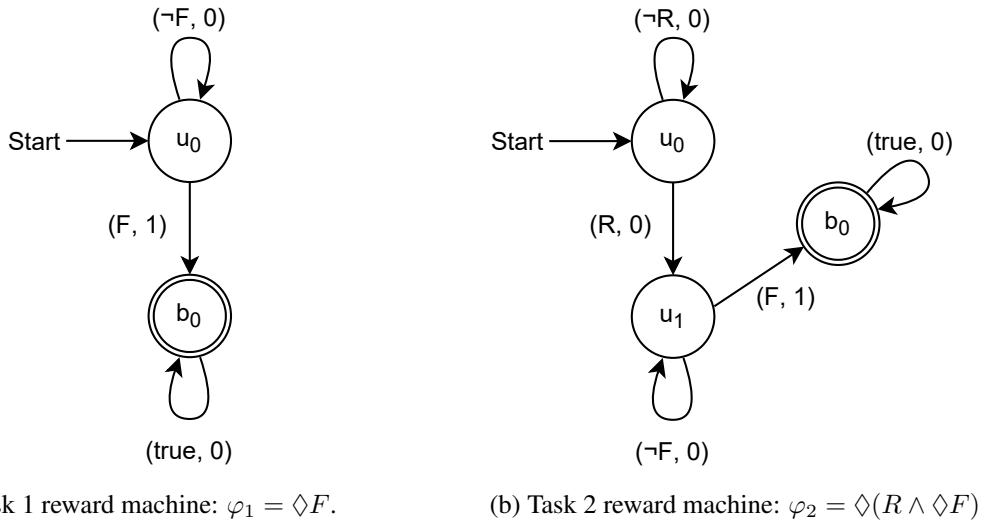


FIGURE 6.4. Simple reward machines for Frozen Lake Tasks 1 and 2. The nodes  $u_0, u_1 \in U$  and  $b_0 \in B$  represent the non-goal-states and goal states of the RM, respectively (goal states are also marked with a double circle). Edges are labelled by the tuple  $(p, r)$  where  $p$  is the logical condition governing the transition, and  $r$  is the reward corresponding to the transition.

By combining these reward machines with the Frozen Lake MDP, we obtain the MDPRMs (Definition 16)  $\mathcal{T}_1$  for Task 1 and  $\mathcal{T}_2$  for Task 2. We use these MDPRMs as input to reinforcement learning algorithms in the following sections.

## 6.3 Optimal Policies

In order to evaluate solutions to the infinite and finite horizon objective problems, we first present the solutions to these problems using dynamic programming (i.e. assuming the environment transition function is known to the agent).

### 6.3.1 Infinite Horizon

We first consider the infinite horizon setting, where the objective is to maximise the probability of saving the friend, regardless of how long it takes. To obtain the optimal values, and thus the optimal policy, we run infinite horizon value iteration (Algorithm 2) on the RMxMDP (Definition 17) corresponding to  $\mathcal{T}_1$ . The output can be seen in Figure 6.5.



Start	1	1	1	1	1
	0.991	0.990	0.991	0.995	1
	0.954	0.945	0.951	0.972	1
	0.791	0.752	0.766	0.856	1
	0	0	0	0	

FIGURE 6.5. Task 1 optimal infinite horizon values. Cell values indicate the probability of task satisfaction from that state.

Since Task 1 has only one reward machine state  $u_0$  excluding the terminal state (Figure 6.4a), we have only one set of optimal values.

The optimal values shown in Figure 6.5 illustrate why, in the infinite horizon setting, one cannot obtain an optimal policy by simply acting greedy with respect to the value function. Such a policy may choose to take a “left” action from a state in the first row, and the policy would never leave that row. As discussed in Section 2.2, to find an optimal policy in this setting, we can assign a non-zero probability to taking all actions that maximise the reward and optimal values in the next step. By doing so, the agent will (eventually) progress along the green path and save the friend. The optimal values also show that the closer the agent gets to the dark blue traps, the less likely it is that the objective is satisfied.

For Task 2, we have two reward machine states  $u_0$  and  $u_1$  excluding the terminal state (Figure 6.4a). Therefore, we have two corresponding sets of optimal values, shown in Figure 6.6.

For the reward machine state  $u_0$  (corresponding to the agent looking for the rope), the agent does best to stay on the safe green path until it can directly transition to the cell containing the rope. Note that

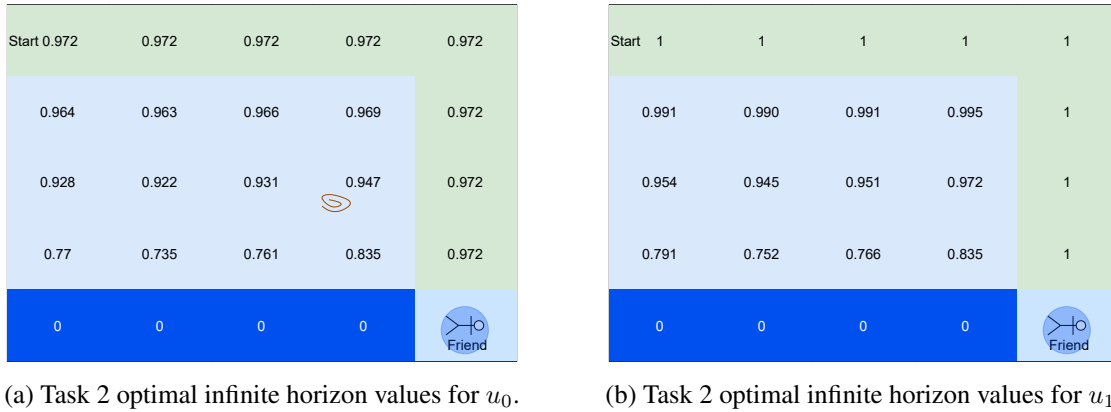


FIGURE 6.6. The optimal infinite horizon environment state values for each reward machine state in Task 2. Note that the rope is only relevant for  $u_0$  (a), as the agent must have already collected the rope to transition to  $u_1$  (b).

while the optimal value for  $u_0$  in the environment state with the rope has a value of 0.947, the value used by the agent during action selection is 0.972: the value of that environment state in  $u_1$ . This is due to the fact that the agent would already have the rope once it enters that state, and can just head towards the friend. For the same reason, we notice that the optimal values for  $u_1$  in Task 2 are identical to the optimal values for  $u_0$  in Task 1 (Figure 6.5).

### 6.3.2 Finite Horizon

For the finite horizon problem, we calculate the optimal values for each task with horizon lengths from 1 to 8. The reason for this is that an optimal policy for a horizon length of 8 will first take an action that maximises the immediate reward plus the optimal value of a state at horizon length 7. The next action will depend on the optimal values at horizon length 6, and so on. The optimal finite horizon values for Task 1 are shown in Figure 6.7.

These figures show that the optimal values increase at every state as the horizon increases (except for the dark blue traps), indicating that the agent has more time and thus more, often safer, options of reaching the goal.



To find a policy that maximises the probability of reaching the goal within  $h$  steps from these figures, first take an action in the direction corresponding to the largest optimal value among adjacent states in the figure for horizon  $h - 1$ . Then, do the same but for horizon  $h - 2$ , and so on. With only one step

remaining (Figure 6.7a), the probability of reaching the goal is zero everywhere except for the two non-trap-states adjacent to the friend. Unlike in the infinite horizon case, we do not have the issue of getting stuck in an infinite loop on the green path when taking actions that maximise the optimal values. To see why, notice that while an optimal policy may take a left action in the top right cell when the horizon is 6, if the agent returns to the top right cell (by taking a right action when the horizon is 5), it will take a down action instead (based on the optimal values in horizon 3).



The optimal values shown in Figure 6.7 were calculated using value iteration, and thus assume knowledge of the transition function in the environment. Since the RL algorithms discussed in this work all converge to optimal Q-values, they share the same optimal values, and thus the optimal policies deduced from the optimal values obtain the same reward. As we will see in Section 6.5, however, some algorithms learn optimal policies with far fewer samples from the environment.

Given that there are two reward machine states in Task 2, and thus double the effective state space of Task 1, we omit the presentation of the optimal values in this work. However, these can be calculated using the code associated with this thesis (<https://github.com/danbraunai/rl-1tl>).





Start	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0.562	1
	0	0	0	0	



(a) Task 1 optimal values with horizon 1.

Start	0	0	0	0	0
	0	0	0	0	0
	0	0	0.316	0.598	1
	0	0	0.316	0.625	1
	0	0	0	0	



(b) Task 1 optimal values with horizon 2.

Start	0	0	0	0	0
	0	0	0.356	0.62	1
	0	0.198	0.409	0.704	1
	0	0.198	0.409	0.764	1
	0	0	0	0	



(c) Task 1 optimal values with horizon 3.

Start	0.178	0.356	0.62	1	1
	0.122	0.264	0.484	0.717	1
	0.135	0.301	0.596	0.847	1
	0.124	0.268	0.524	0.783	1
	0	0	0	0	



(d) Task 1 optimal values with horizon 4.

Start	0.356	0.62	1	1	1
	0.278	0.484	0.794	0.909	1
	0.243	0.455	0.685	0.881	1
	0.218	0.403	0.598	0.783	1
	0	0	0	0	



(e) Task 1 optimal values with horizon 5.

Start	0.620	1	1	1	1
	0.497	0.777	0.877	0.96	1
	0.403	0.628	0.788	0.925	1
	0.337	0.480	0.643	0.823	1
	0	0	0	0	

(f) Task 1 optimal values with horizon 6.

Start	1	1	1	1	1
	0.807	0.863	0.942	0.974	1
	0.634	0.739	0.862	0.943	1
	0.472	0.569	0.690	0.823	1
	0	0	0	0	

(g) Task 1 optimal values with horizon 7.

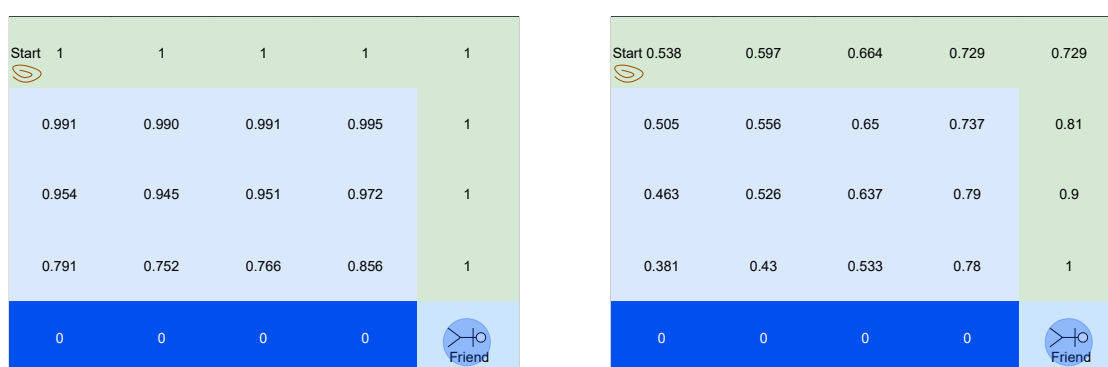
Start	1	1	1	1	1
	0.916	0.937	0.961	0.984	1
	0.766	0.836	0.897	0.956	1
	0.579	0.643	0.718	0.835	1
	0	0	0	0	

(h) Task 1 optimal values with horizon 8.

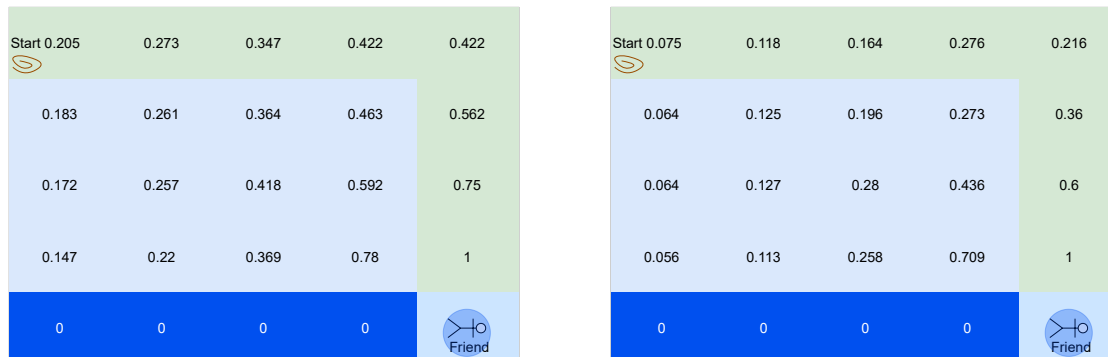
FIGURE 6.7. The optimal finite horizon values obtained from value iteration for Task 1 with  $\gamma = 1$ .

## 6.4 Heuristic Methods

As is commonly done in the reward-based RL literature, one can use an infinite horizon algorithm and reduce the discount factor  $\gamma$  in order to force the agent to collect rewards more quickly. In an objective-based RL setting, this can correspond to completing the objective more quickly. In Figure 6.8 we run infinite horizon value iteration on Task 1 with  $\gamma \in [1, 0.9, 0.75, 0.6]$ .



(a) Task 1 optimal infinite horizon values with  $\gamma = 1$ . (b) Task 1 optimal infinite horizon values with  $\gamma = 0.9$ .



(c) Task 1 optimal infinite horizon values with  $\gamma = 0.75$ . (d) Task 1 optimal infinite horizon values with  $\gamma = 0.6$ .

FIGURE 6.8. The optimal infinite horizon values for Task 1 with different settings for the discount factor  $\gamma$ .

With  $\gamma = 1$  (shown also in Figure 6.5), we obtain the “safe” policy following the green path. With each reduction in  $\gamma$ , the optimal agent enters the slippery ice earlier, increasing the risk of falling in the trap while also increasing the probability of reaching the friend more quickly. With  $\gamma = 0.9$ , the optimal policy takes two steps on the green path before moving down-right to a slippery grid. The  $\gamma = 0.75$  agent enters the ice after only one step on the green path, and the  $\gamma = 0.6$  enters the ice immediately.

As discussed in Section 2.2, with  $\gamma < 1$ , we can obtain the optimal policy by acting greedily with respect to the optimal values. This property is illustrated in these figures, where, when  $\gamma < 1$ , there do not exist looping paths when taking actions that maximise the values of adjacent squares.

To see why step-based penalties do not work in general for infinite horizon MDPs, consider the optimal values shown in Figure 6.9 with  $\gamma = 0.99$  and a step-based penalty of 0.1. As the figure shows, the value


Start 0.296	0.397	0.497	0.598	0.598
-0.639	-0.740	-0.491	-0.059	0.698
-4.534	-5.308	-4.444	-2.102	0.799
-20.927	-24.685	-23.020	-13.764	0.900
-99.990	-99.990	-99.990	-99.990	

FIGURE 6.9. Task 1 optimal infinite horizon values with step based penalties of 0.1 and  $\gamma = 0.999$ .

of the states corresponding to traps are very small. This is due to the fact that the agent will continue to receive a reward of  $-0.1$  for every future timestep in perpetuity. This of course has the effect of lowering the optimal values of all other state which have a non-zero probability of entering a trap (i.e. any ice state). If we instead set  $\gamma = 1$ , these values would be infinite and value iteration would not converge.

Another way around the issue of value iteration not converging, apart from setting  $\gamma < 1$ , is to set a horizon in which the agent can no longer receive rewards and where the episode is restarted. While this can allow for step-based penalties which encourage the agent to collect rewards faster, the magnitude of the horizon has a large influence on how quickly the agent is incentivised to collect rewards. As in the case of adjusting the discount factor, this parameter can be hard for the designer to set and achieve the desired result without trial and error.

Of course, as shown in Section 5.3, one cannot obtain an optimal policy for an arbitrary finite horizon MDP using the methods above, as an optimal policy may be time-dependent and not memoryless.

## 6.5 Model Efficiency Experiments

In this section, we compare the sample and update efficiency of Q-learning with CRM (Algorithm 4) in the infinite horizon setting, as well as Q-learning with QT-learning, QRM-learning and QTRM-learning (Algorithm 5) in the finite horizon setting. Note that we use the words *sample* and *step* to refer to a single  $(s, a, s')$  experience.

In the infinite horizon setting, each episode begins in the start state in the top left corner of the grid, and ends either when the agent completes the task, falls in a trap, or takes longer than 50 steps. We set a reward discount factor of  $\gamma = 0.99$  in order to avoid learning policies that get stuck in infinite loops (see end of Section 2.2 and Section 6.3.1). We set  $\alpha = 0.01$  for the initial learning rate, which decays by a factor of 0.9 every 200000 steps.

In the finite horizon setting, we set the `episodes_per_reset` to  $N = 3$ , meaning that every third episode begins in the initial state in the top left corner of the grid, with the other two episodes beginning with the agent in the same environment and RM state as it ended the previous episode (see end of Section 5.2 for how this improves training). Each episode ends when the agent completes the task, falls in a trap, or reaches the fixed time horizon. No reward discounting is used ( $\gamma = 1$ ). The initial learning rate  $\alpha$  is set to 0.1, decaying at a rate of 0.9 every 100000 steps<sup>1</sup>.

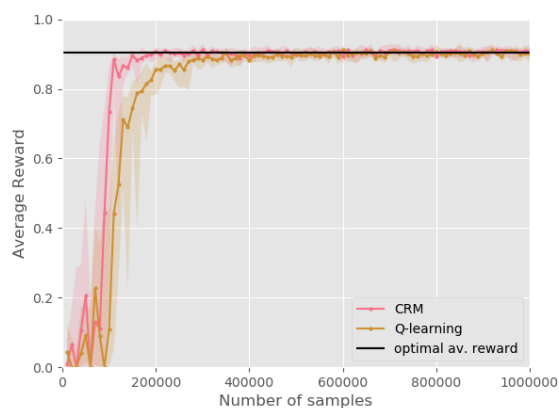
For both settings, we set  $\epsilon = 0.1$  so that the agent takes a random action (not the action suggested by the current Q-values) 10% of the time. To encourage exploration, we initialised the Q-values *optimistically* by setting all values to 1.0001. We run each algorithm for 30 independent trials. Every 10000 steps, we take a snapshot of the Q-values and measure the average reward over 50 episodes when following a policy that acts greedily with respect to these Q-values. We then take the median of these average rewards over the 30 independent trials to produce the *average reward* metric used throughout this section. The average reward is plotted against the number of samples and the number of Q-value updates used in training. Since Task 1 has only a single reward machine state and thus does not generate counterfactual experiences for multiple reward machine states, we use Task 2 and Task 3 for these comparisons.

The results for the infinite horizon setting in Task 2 is shown in Figure 6.10.

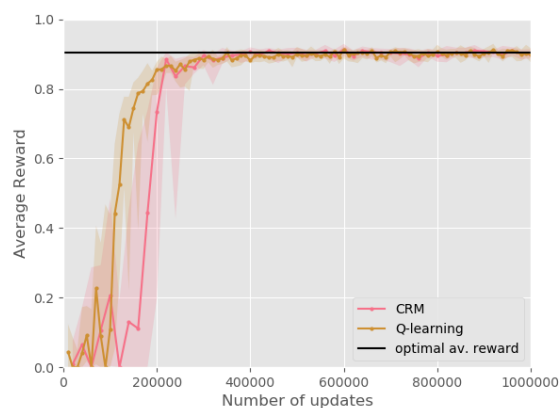
From Figure 6.10a, we can see that CRM converges to an optimal policy with fewer samples than Q-learning in Task 2, although the difference is not significant. In Task 3 the difference in sample efficiency

---

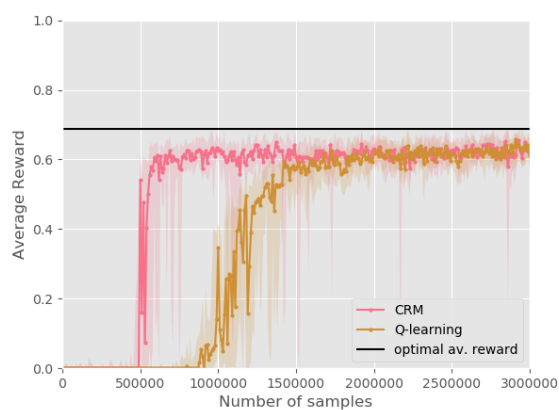
<sup>1</sup>We found that the finite horizon policies were less erratic during training and could handle a larger learning rate.



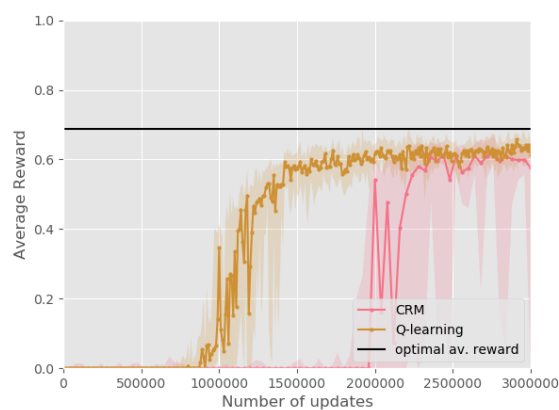
(a) Task 2 reward vs. samples.



(b) Task 2 reward vs. Q updates.



(c) Task 3 reward vs. samples.



(d) Task 3 reward vs. Q updates.

FIGURE 6.10. Infinite horizon sample and update efficiency for Tasks 2 and 3 with  $\gamma = 0.99$ . The black line represents the optimal average reward (obtained using value iteration). Shaded regions represent the average reward of the 25% and 75% quartiles over the 30 independent runs.

is more pronounced, with CRM learning a good policy with almost half the amount of samples. This result should perhaps not be surprising: in Task 2, the most useful experiences for when the agent has the rope (i.e. in RM state  $u_1$ ) will be those in the bottom right corner of the grid. When the agent doesn't have the rope (i.e. in RM state  $u_0$ ), it is unlikely to collect experiences in the bottom right corner, and so most of the counterfactual experiences will not be valuable. On the other hand, in Task 3 where the ropes are scattered across the grid, the experiences collected in any RM state are more likely to be useful in other RM states, thus getting more value out of the counterfactual experiences.

With regard to update efficiency, the reverse pattern can be seen; Q-learning finds good policies with fewer updates than CRM, with a more pronounced difference for Task 3. Recall from Algorithm 4 that

Q-learning does one update per sample, whereas CRM will run  $|U|$  updates per sample (where  $|U| = 2$  RM states for Task 2 and  $|U| = 4$  RM states for Task 3).

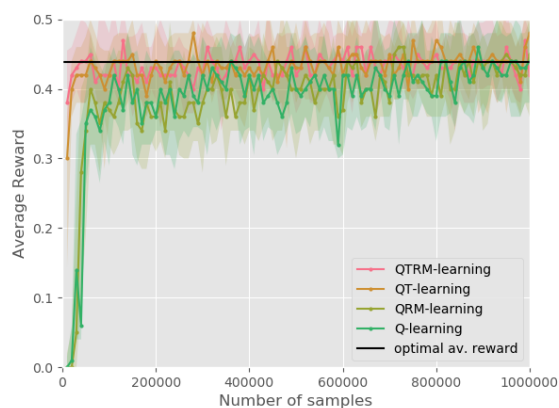
These figures show that the counterfactual updates in CRM are useful for the agent (since the average reward is higher per sample), but not as useful as true experiences (since the average reward is lower per Q-value update). Notice that, before the learned policies get close to optimal, there is a lot of noise in the average reward. This is a problem that commonly affects RL algorithms that choose actions  $\epsilon$ -greedily based on value functions, and occurs when minor changes to values cause a large change to a policy. This is especially the case for the infinite horizon problem, where a change in policy at a state determines the agent’s actions at that state for all timesteps. In the finite horizon problem, a change in policy at a (timestep, state) pair only affects the policy at that timestep, and we will soon see that the corresponding figures are smoother.

The algorithm comparisons for Tasks 2 and 3 in the finite horizon setting are shown in Figure 6.11 (see Figures 2.1 and 2.2 in the Appendix for versions of the sample efficiency figures zoomed-in on the early stages of learning).

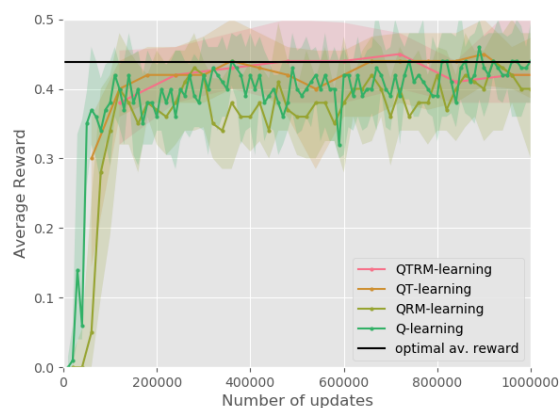
Figure 6.11a shows that algorithms that use counterfactual experiences for each timestep (QTRM-learning and QT-learning) in Task 2 learn effective policies with fewer samples than those that don’t (QRM-learning and Q-learning). With regard to the value of counterfactual experiences for RM states, the difference in sample efficiency between QTRM-learning (resp. QRM-learning) and QT-learning (resp. Q-learning) is insignificant, indicating that the agent benefits more from counterfactual experiences for each timestep than for RM states. For Task 3 (Figure 6.11c), the difference in sample efficiency between the algorithms is much more prominent, with QTRM-learning achieving the best sample efficiency, closely followed by QT-learning. Despite displaying worse sample efficiency than those that use counterfactual experiences for each timestep, QRM-learning converges to an optimal policy much faster than Q-learning. In fact, it takes Q-learning  $\sim 500000$  samples to learn a policy that achieves any reward. The reason for this is that, in order for Q-learning to propagate non-zero rewards from completing the task throughout the set of Q-values, it is not enough for it satisfy the complex objective in 45 timesteps ( $N \times H$ ) while acting almost randomly<sup>2</sup> just once. In order for a reward to influence the policy at more than one state, the agent must subsequently find a transition to the exact (environment and RM) state

---

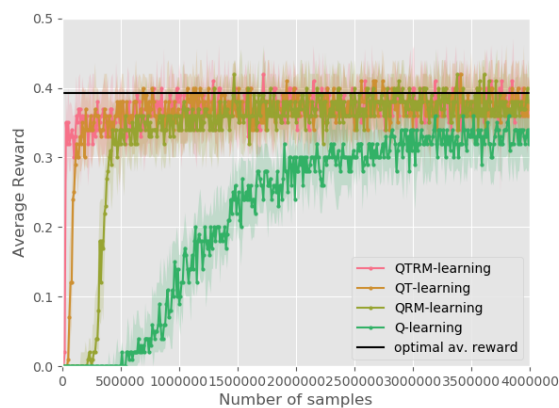
<sup>2</sup>Since we use optimistic initialisation, actions which do not lead to rewards will be disincentivised, and so the agent doesn’t act truly randomly before it finds its first reward.



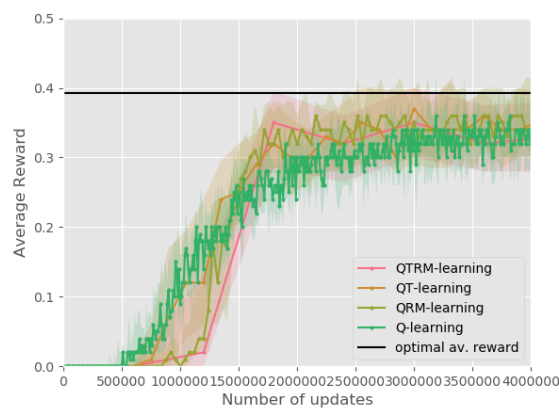
(a) Task 2 reward vs. samples.



(b) Task 2 reward vs. Q updates.



(c) Task 3 reward vs. samples.



(d) Task 3 reward vs. Q updates.

FIGURE 6.11. Finite horizon sample and update efficiency with  $\gamma = 1$  (no discounting). Task 2 has a horizon of 6. Task 3 has a horizon of 15. Shaded regions represent the average reward of the 25% and 75% quartiles over the 30 independent runs.

and at the preceding timestep in which it previously transitioned to the rewarding state. With counterfactual experiences, such stringent conditions are not required and non-zero rewards can propagate faster throughout the set of Q-values.

The difference in update efficiency between the algorithms is less clear than in the infinite horizon case.<sup>3</sup> For Task 2, all algorithms present a similar update efficiency, with large variability in the estimates. In Task 3, Q-learning is able to start learning rewarding policies with fewer updates than the other algorithms, but then becomes less update efficient once it can solve the task  $\sim 0.25\%$  of the time.

<sup>3</sup>Note that, since QTRM-learning performs  $|U| \times H$  updates per sample, and since our policies are evaluated every 10000 samples, we have fewer data points for QTRM-learning compared to other algorithms.

Overall, these results are encouraging, showing that QTRM-learning is able to learn good policies with fewer samples than the baseline methods, and that the counterfactual experiences used to update the policy are not clearly less valuable than real experiences.

## 6.6 Code

The code used in this thesis is available at <https://github.com/danbraunai/rl-1tl>. The repository includes the Frozen Lake environment, integrated within the OpenAI gym API (Brockman et al. 2016), along with all algorithms and experiments presented in this thesis.



## Discussion

---

In this chapter, we discuss various aspects of the methods proposed in the preceding sections, including limitations, and potential directions for future work.

### 7.1 Terminal States

As mentioned in Section 6.1, there is a trade-off between expressing the terminal nature of the dark blue traps in the temporal logic formula (with the use of  $\Box\neg T$ ), or in the environment itself by enforcing self-looping transitions and no future rewards. While these constraints express the same property, they can have different consequences on the learning efficiency of RL algorithms. Adding an extra term to the formula requires an additional state in the reward machine to express it. This means that the QRM-learning and QTRM-learning algorithms must update the Q-value for an extra state of the reward machine with every transition in the environment. This problem could be remedied by preventing the learning algorithm from making such updates for this particular reward machine state, but this would require extra work by the task designer. By instead enforcing self-looping transitions in the dark blue traps, learning algorithms will, by default, compute unproductive updates for every self-loop transition when an agent enters that state. To avoid this, the standard approach (adopted in this work) is to output a binary *done* indicator whenever the agent reaches any terminal state, such as a dark blue trap or when an objective has been completed. In general, when learning efficiency is not paramount, expressing all terminal conditions in the formula is the more natural approach and requires the least amount of effort for the task designer.

## 7.2 Other RL Paradigms

While the methods presented in Chapter 5 are all based on tabular Q-learning, counterfactual experience methods can also be used in off-policy RL algorithms that use function approximation such as neural networks. In Section 4.5, we showed how Icarte et al. 2020 adapted the Deep Q-learning (DQN) and Deep Deterministic Policy Gradient (DDPG) algorithms can be improved by adding counterfactual experiences to the experience replay buffer. While the authors added counterfactual experiences for each RM state, the same method is likely to be effective when also adding counterfactual experiences for each timestep, as in QTRM-learning.

By definition, counterfactual experiences only apply to off-policy algorithms, as the experiences used to update the policy in counterfactual states do not come from the current policy (since the agent may have taken different actions in different RM states). However, it may be possible to benefit from counterfactual experiences in (classically) on-policy algorithms such as the actor-critic family (A2C, A3C) and Proximal Policy Optimization (PPO) with the use of “importance sampling” (Sutton and Barto 2018).

## 7.3 Improving Update Efficiency

Many off-policy RL algorithms, including those discussed in this work, store experiences in an experience replay buffer. Given that some (true or counterfactual) experiences are more likely to help learn a good policy, one can improve learning efficiency by sampling the more useful experiences more often from the experience replay buffer, as opposed to sampling uniformly or exhaustively (as is done in this work). Although methods such as *prioritized experience replay* (Schaul et al. 2015) address this by sampling experience with higher temporal difference error more frequently, this is unlikely to be effective for counterfactual experiences that are very unlikely to occur under a good policy, but have high temporal difference errors. One promising avenue for future work is to sample experiences according to the likelihood of that experience occurring in a real environment rollout. As the agent gathers experiences, the likelihood of each experience is updated and sampling takes place on the updated likelihoods.

Not only can it be important to prioritise the sampling of certain transitions in an experience replay buffer, but controlling the number of counterfactual experiences stored in the first place can also reduce the amount of computation required. Creating and updating Q-values for counterfactual experiences corresponding to every timestep in a finite horizon task can be computationally expensive, particularly

for long time horizons. A method to address this problem would be to store only a random sample of these counterfactual experiences. When choosing an action, if the Q-values corresponding to the current timestep were only created from few experiences, the Q-values of neighbouring timesteps could be used to select an ( $\epsilon$ -greedy) action. The convergence guarantees for such an algorithm would be maintained as all experiences would be sampled an infinite number of times, eventually.

Learning from counterfactual experiences in QT-learning and QTRM-learning can be more robust to changes in the initial state distribution. Recall from the end of Section 2.1 that Q-learning based methods converge to a policy that maximises the expected reward from any starting state in the MDP. However, in practice, the learned policies of Q-learning are mostly optimised on trajectories that are common given the initial state distribution. In a setting where the initial state distribution changes, or where there are multiple tasks to be learned via transfer learning, counterfactual experiences for each timestep will allow the agent to learn a good policy on states that are common in particular timesteps for one initial state distribution or task, even when it is in another initial state distribution or task.

## 7.4 Reward Shaping

The reward machine literature has shown that reward shaping (Section 4.2) produces mixed results in overcoming reward sparsity in a variety of environments and tasks in the infinite horizon setting (Camacho, Icarte, et al. 2019; Icarte et al. 2020). Future work would explore the use of reward shaping in the finite horizon setting for objective-based RL. In this setting, extra rewards would be given for each state in the reward machine, as well as timesteps leading up the horizon. To create these rewards, a potential function could be defined on an extended reward machine that contains a unique state for each (timestep, RM state) pair. In practice, the resulting shaped rewards may look similar to a step-based penalty with additional rewards defined on transitions to new RM states (in the original RM). Reward shaping methods could improve the convergence rate of RL algorithms in the finite horizon objective-based RL setting, and allow for methods to scale to much larger environments and complex objectives.

## 7.5 Time-Dependent and Reward-Machine-Dependent Environment Transitions

The methods discussed in this work assume that the environment transitions are independent of both the reward machine state and the current timestep. Realistic scenarios may breach these conditions. For example, in the slippery gridworld environment, the agent may be less likely to slip once it has collected the rope, as it can use it as a counterbalance when falling. In this case, the environment transitions are dependent on the current reward machine state (indicating whether the agent has the rope or not). Of course, one can simply extend the state space using the reward machine states and create a new transition function dependent only on the new state space, but more efficient techniques may be available using the structure of the reward machine.

In the situation where environment transitions are dependent on the timestep, counterfactual experience methods such as QT-learning and QTRM-learning are inapplicable. This is due to the fact that, given an action, an environment transition sampled from one timestep does not correspond to a randomly sampled transition in another timestep. Therefore, the counterfactual experiences over timesteps will not represent valid data on which to update the Q-values. While vanilla Q-learning will converge to an optimal policy in this setting, methods to speed up learning by making use of temporal structure are left for future work.

## 7.6 Minimising Time

The problems in this work are focused on maximising the probability of completing a task within a fixed deadline. Another useful problem setting is one which aims to minimise the expected time taken to complete a task, conditioned on the task being completed at least a fixed percentage of the time. Motivating real-world problems in this setting include “manufacture these items as quickly as possible, ensuring that they pass quality control at least 90% of the time”, or “drive me to work as quickly as possible, ensuring that we do not get in an accident more than 0.01% of the time”. The special case of this problem where the policy must guarantee task completion (the minimum task completion percentage is 100%), is known as the Stochastic Shortest Path (SSP) problem, and is well studied in the optimal control literature (Randour et al. 2015).<sup>1</sup> The problem of finding worst-case policies, those that maximise

---

<sup>1</sup>In fact, the SSP problem finds a policy that minimises an expected cost, where a cost is associated with each transition. This is more general than minimising the expected time, which is equivalent to setting the cost of each transition to  $-1$ .

the expected time conditioned on the task being completed at all (i.e.  $\geq 0\%$  of the time), is studied in C. Baier, Klein, et al. 2017. As well as addressing a different problem (maximising expected time as opposed to minimising), the methods proposed by the authors rely on having access to the environment transition function. Nonetheless, the work provides useful insights into the nature of these types of problems, and could lay the foundation for a new suite of reinforcement algorithms that learn policies which minimise or maximise the conditional expectations of satisfying tasks expressed in temporal logic.

## Conclusion

---

This research sought to analyse and extend the toolkit available to reinforcement learning practitioners for problems expressible in variants of Linear Temporal Logic. We showed that heuristic methods, such as step-based penalties and reward discounting, can be effective in learning policies that collect rewards quickly, but may be suboptimal for objectives that must be completed before a fixed deadline. We then introduced QTRM-learning, a novel extension of the Q-learning-based methods reported in the reward machines literature. QTRM-learning takes advantage of the independence between transitions in a Markov Decision Process, states of a reward machine, and timesteps, in order to learn policies that maximise the probability of satisfying a temporal logic formula before a deadline. For tasks of varying difficulty defined in a slippery gridworld, we showed that QTRM-learning can find effective policies with far fewer samples than more naive Q-learning baselines. Moreover, the experiments show that the additional updates used in QTRM-learning, obtained from counterfactual experiences, can be just as beneficial to the agent as those obtained from real experiences. With many opportunities for further refinement, methods such as QTRM-learning take a step towards meeting the demand for intelligent systems that can solve complex, real-world tasks under rigid constraints.

## Bibliography

- Abbeel, Pieter and Andrew Y Ng (2005). “Exploration and apprenticeship learning in reinforcement learning”. In: *Proceedings of the 22nd international conference on Machine learning*, pp. 1–8.
- Amodei, Dario et al. (2016). “Concrete Problems in AI Safety”. In: arXiv: 1606.06565 [cs.AI].
- Andrychowicz, OpenAI: Marcin et al. (2020). “Learning dexterous in-hand manipulation”. In: *The International Journal of Robotics Research* 39 (1), pp. 3–20. DOI: 10.1177/0278364919887447. URL: <https://doi.org/10.1177/0278364919887447>.
- Baier, Christel and Joost-Pieter Katoen (2008). *Principles of Model Checking (Representation and Mind Series)*. The MIT Press. ISBN: 026202649X.
- Baier, Christel, Joachim Klein, et al. (2017). “Maximizing the conditional expected reward for reaching the goal”. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 269–285.
- Baier, Jorge A and Sheila A McIlraith (2006). “Planning with Temporally Extended Goals Using Heuristic Search.” In: *ICAPS*, pp. 342–345.
- Bellman, Richard (1966). “Dynamic programming”. In: *Science* 153 (3731), pp. 34–37.
- Bienvenu, Meghyn et al. (2006). “Planning with Qualitative Temporal Preferences.” In: *KR* 6, pp. 134–144.
- Bozkurt, A K et al. (2020). “Control Synthesis from Linear Temporal Logic Specifications using Model-Free Reinforcement Learning”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10349–10355. DOI: 10.1109/ICRA40945.2020.9196796.
- Bozkurt, Alper Kamil et al. (2021). “Learning Optimal Strategies for Temporal Tasks in Stochastic Games”. In: *arXiv preprint arXiv:2102.04307*.
- Brockman, Greg et al. (2016). “Openai gym”. In: *arXiv preprint arXiv:1606.01540*.
- Camacho, Alberto, Rodrigo Toro Icarte, et al. (2019). “LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning.” In: *IJCAI* 19, pp. 6065–6073.
- Camacho, Alberto and Sheila A McIlraith (2019). “Learning interpretable models expressed in linear temporal logic”. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 29, pp. 621–630.
- Chou, Glen et al. (2020). “Explaining Multi-stage Tasks by Learning Temporal Logic Formulas from Suboptimal Demonstrations”. In: *arXiv preprint arXiv:2006.02411*.
- Gabaldon, Alfredo (2004). “Precondition Control and the Progression Algorithm.” In: *ICAPS*, pp. 23–32.

- Garca, Javier and Fernando Fernández (2015). “A comprehensive survey on safe reinforcement learning”. In: *Journal of Machine Learning Research* 16 (1), pp. 1437–1480.
- Giacomo, Giuseppe De, Luca Iocchi, et al. (2019). “Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications”. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 29, pp. 128–136.
- Giacomo, Giuseppe De and Moshe Y Vardi (2013). “Linear temporal logic and linear dynamic logic on finite traces”. In: *IJCAI’13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pp. 854–860.
- Hahn, Ernst Moritz et al. (2019). “Omega-regular objectives in model-free reinforcement learning”. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 395–412.
- Harada, Daishi (1997). “Reinforcement learning with time”. In: *AAAI/IAAI*, pp. 577–582.
- Hasanbeig, Mohammadhosein, Alessandro Abate, et al. (2018). “Logically-constrained reinforcement learning”. In: *arXiv preprint arXiv:1801.08099*.
- (2020a). “Cautious reinforcement learning with logical constraints”. In: *arXiv preprint arXiv:2002.12156*.
- (2020b). “Certified Reinforcement Learning with Logic Guidance”. In: *arXiv preprint arXiv:1902.00778*.
- Hasanbeig, Mohammadhosein, Yiannis Kantaros, et al. (2019). “Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees”. In: *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 5338–5343.
- Hasselt, Hado Van et al. (2016). “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI conference on artificial intelligence* 30 (1).
- Icarte, Rodrigo Toro et al. (2018a). “Teaching multiple tasks to an RL agent using LTL”. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 452–461.
- (2018b). “Using reward machines for high-level task specification and decomposition in reinforcement learning”. In: *International Conference on Machine Learning*, pp. 2107–2116.
- (2020). “Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning”. In: *arXiv preprint arXiv:2010.03950*.
- Kearns, Michael and Satinder Singh (2002). “Near-optimal reinforcement learning in polynomial time”. In: *Machine learning* 49 (2), pp. 209–232.
- Kupferman, Orna and Moshe Y Vardi (2001). “Model checking of safety properties”. In: *Formal Methods in System Design* 19 (3), pp. 291–314.
- Kwiatkowska, Marta et al. (2011). “PRISM 4.0: Verification of probabilistic real-time systems”. In: *International conference on computer aided verification*, pp. 585–591.
- Lacerda, Bruno et al. (2015). “Optimal Policy Generation for Partially Satisfiable Co-Safe LTL Specifications.” In: *IJCAI*, pp. 1587–1593.
- Li, X et al. (2017). “Reinforcement learning with temporal logic rewards”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3834–3839. DOI: 10.1109/IROS.2017.8206234.
- Li, Yuxi (2017). “Deep reinforcement learning: An overview”. In: *arXiv preprint arXiv:1701.07274*.



- Lillicrap, Timothy P et al. (2015). “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971*.
- Littman, M et al. (2017). “Environment-Independent Task Specifications via GLTL”. In: *ArXiv abs/1704.04341*.
- Mnih, Volodymyr et al. (May 2015). “Human-level control through deep reinforcement learning”. In: *Nature* 518 (7540), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: <https://doi.org/10.1038/nature14236>.
- Ng, Andrew Y, Daishi Harada, et al. (1999). “Policy invariance under reward transformations: Theory and application to reward shaping”. In: *Icml 99*, pp. 278–287.
- Ng, Andrew Y, Stuart J Russell, et al. (2000). “Algorithms for inverse reinforcement learning.” In: *Icml* 1, p. 2.
- Oura, Ryohei et al. (2020). “Reinforcement learning of control policy for linear temporal logic specifications using limit-deterministic generalized B"uchi automata”. In: *IEEE Control Systems Letters* 4 (3), pp. 761–766.
- Pardo, Fabio et al. (2018). “Time limits in reinforcement learning”. In: *International Conference on Machine Learning*, pp. 4045–4054.
- Pnueli, Amir (1977). “The temporal logic of programs”. In: *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS 1977-October*, pp. 46–57. ISSN: 02725428. DOI: 10.1109/sfcs.1977.32.
- Puterman., M. L. (1994). *Markov Decision Processes*. Wiley.
- Rabin, Michael O and Dana Scott (1959). “Finite automata and their decision problems”. In: *IBM journal of research and development* 3 (2), pp. 114–125.
- Randour, Mickael et al. (2015). “Variations on the stochastic shortest path problem”. In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pp. 1–18.
- Sadigh, D et al. (2014). “A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications”. In: *53rd IEEE Conference on Decision and Control*, pp. 1091–1096. DOI: 10.1109/CDC.2014.7039527.
- Safra, S (1988). “On the complexity of omega -automata”. In: *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pp. 319–327. DOI: 10.1109/SFCS.1988.21948.
- Schaul, Tom et al. (2015). “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952*.
- Schulman, John et al. (2017). “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347*.
- Shah, Ankit Jayesh et al. (2018). “Bayesian inference of temporal task specifications from demonstrations”. In:
- Silver, David, Aja Huang, et al. (2016). “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (7587). ISSN: 14764687. DOI: 10.1038/nature16961.
- Silver, David, Satinder Singh, et al. (2021). “Reward is enough”. In: *Artificial Intelligence*, p. 103535.
- Sistla, A Prasad and Edmund M Clarke (1985). “The complexity of propositional linear temporal logics”. In: *Journal of the ACM (JACM)* 32 (3), pp. 733–749.
- Strehl, Alexander L et al. (2006). “PAC model-free reinforcement learning”. In: *Proceedings of the 23rd international conference on Machine learning*, pp. 881–888.

- Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.
- Wang, Chuanzheng et al. (2020). *Continuous Motion Planning with Temporal Logic Specifications using Deep Neural Networks*.
- Watkins, Christopher J C H and Peter Dayan (1992). “Q-learning”. In: *Machine learning* 8 (3-4), pp. 279–292.
- Wells, Andrew M et al. (2020). “LTLf Synthesis on Probabilistic Systems”. In: *arXiv preprint arXiv:2009.10883*.
- Wilke, Thomas (1999). “Classifying discrete temporal properties”. In: *Annual symposium on theoretical aspects of computer science*, pp. 32–46.
- Wilson, Aaron et al. (2007). “Multi-task reinforcement learning: a hierarchical bayesian approach”. In: *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022.

## Appendix

### 1 Additional Background

To define probability spaces on infinite trajectories (Definition 8), we make use of  $\sigma$ -algebras:

DEFINITION 18 ( $\sigma$ -Algebra). *Given a set  $X$ , a  $\sigma$ -algebra  $F$  is a non-empty collection of subsets of  $X$  such that:*

- (1)  $X \in F$ ,
- (2)  $F$  is closed under countable unions: If  $A_i \in F$  for  $i \in \mathbb{N}$ , then  $\bigcup_{i=1}^{\infty} A_i \in F$ ,
- (3)  $F$  is closed under complements: If  $A \in F$  then  $X \setminus A \in F$ .

### 2 Additional Figures

To better distinguish between the sample efficiency of the finite horizon algorithms presented in this work for Task 2, Figure 2.1 shows a version of Figure 6.11a zoomed in on the first 100000 learning samples.

Similarly, Figure 2.2 shows a version of Figure 6.11c zoomed in on the first 1000000 learning samples.

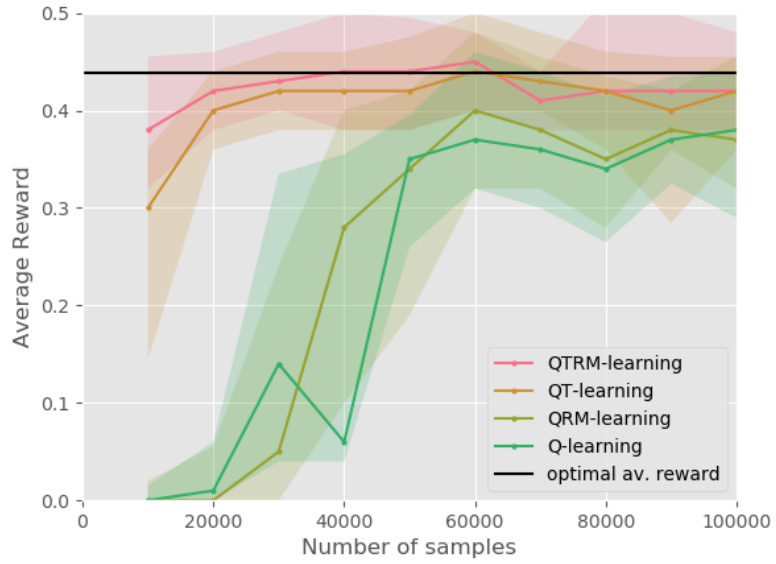


FIGURE 2.1. Task 2 reward vs. samples.

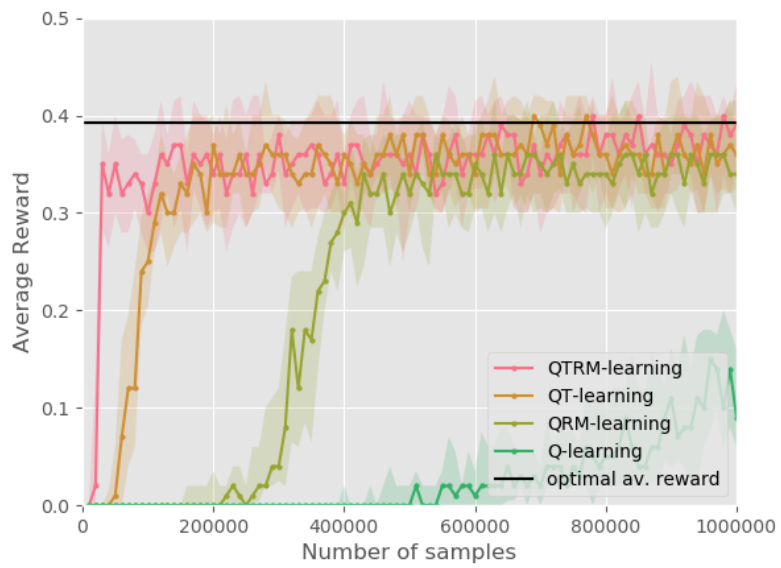


FIGURE 2.2. Task 3 reward vs. samples.