On the Learnability of General Reinforcement-learning Objectives

by

Cambridge Yang

B.S, University of California, Berkeley (2017)
S.M., Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

Authored by: Cambridge Yang
Department of Electrical Engineering and Computer Science
February 28, 2025

Certified by: Michael Carbin
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

On the Learnability of General Reinforcement-learning Objectives

by

Cambridge Yang

Submitted to the Department of Electrical Engineering and Computer Science
on February 28, 2025 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Reinforcement learning enables agents to learn decision-making policies in unknown environments to achieve specified objectives. Traditionally, these objectives are expressed through reward functions, enabling well-established guarantees on learning near-optimal policies with a high probability — a property known as probably approximately correct (PAC) -learnability. However, reward functions often serve as imperfect surrogates for true objectives, leading to reward hacking and undermining these guarantees.

This thesis formalizes the specification and learnability of general reinforcement-learning objectives beyond rewards, addressing fundamental questions of expressivity and policy learnability. I examine three increasingly expressive classes of objectives: (1) Linear Temporal Logic (LTL) objectives, which extend conventional scalar rewards to temporal specifications of behavior and have garnered recent attention, (2) Computable objectives, encompassing a broad class of structured, algorithmically definable objectives and (3) Non-computable objectives, representing general objectives beyond the computable class.

For LTL objectives, I prove that only finitary LTL objectives are PAC-learnable, while infinite-horizon LTL objectives are inherently intractable under the PAC-MDP framework. Extending this result, I establish a general criterion: an objective is PAC-learnable if it is continuous and computable. This criterion facilitates the establishment of PAC-learnability for various existing classes of objectives with unknown PAC-learnability and informs the design of new, learnable objective specifications. Finally, for non-computable objectives, I introduce limit PAC-learnability, a practical relaxation where a sequence of computable, PAC-learnable objectives approximates a non-computable objective. I formalize a universal representation of non-computable objectives using nested limits of computable functions and provide sufficient conditions under which limit PAC-learnability holds.

By establishing a theoretical foundation for general RL objectives, this thesis advances our understanding of which objectives are learnable, how they can be specified, and how agents can effectively learn policies to optimize them. These results contribute to the broader goal of designing intelligent agents that align with expressive, formally defined objectives—moving beyond the limitations of reward-based surrogates.

Thesis supervisor: Michael Carbin
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

This dissertation would not have been possible without the guidance, support, and encouragement of many individuals I am deeply grateful to.

First and foremost, I would like to express my sincere gratitude to my advisor, Michael Carbin, for his invaluable mentorship, insightful discussions, and unwavering support throughout my PhD journey. His guidance has been instrumental in shaping my research, my writing, and my thinking. I greatly appreciate his dedication, thoughtfulness, and effort in managing the research group I was fortunate to be a part of.

I am also immensely thankful to my thesis committee members, Leslie Kaelbling, Michael Littman, and Armando Solar-Lezama, for their encouragement, thoughtful feedback, and inspiring discussions that helped refine and strengthen this research.

A special acknowledgment goes to Michael Littman, whose mentorship and collaboration have been foundational in this work. This research would not have existed without the collaboration we started six years ago, and I am deeply grateful for the insights and inspiration he has provided along the way.

I would also like to extend my appreciation to all current and past members of our research group, as well as friends and colleagues who have provided support, insightful discussions, and a strong sense of community throughout my PhD.

The research presented in this thesis would not have been possible without the support of the following funding sources: the Office of Naval Research (ONR) under grant N00014-17-1-2699, the National Science Foundation (NSF) under Award No. 1918839, the Sloan Foundation and SRC JUMP 2.0, and the Defense Advanced Research Projects Agency (DARPA) under Award No. HR001118C0059. I gratefully acknowledge their support.

To my family and friends, I am deeply grateful for your love, patience, and encouragement. Your unwavering belief in me has been a constant source of strength. Above all, I thank Dee — my partner and now wife — whose love and support have been unshakable. This journey has been ours together, and I could not have done it without her.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Intelligent agents are increasingly prevalent across diverse fields. An *intelligent agent* is a system that observes its environment, makes inferences about it, and takes actions to optimize its *objective* [3]. They respond to user queries in conversational systems [4], achieve superhuman performance in games [5], aid in drug discovery by predicting protein structures [6], identify treatments in healthcare [7], schedule tasks in data centers [8], and operate robots for manufacturing, driving, and household tasks [9].

Reinforcement learning is one well-established technique for constructing an intelligent agent systems. In reinforcement learning, we situate an autonomous agent in an unknown environment and specify an objective. The objective for the agent is a specification over possible trajectories of the overall system — the environment and the agent. Each *trajectory* is an infinite sequence of the states of the system, evolving through time. The objective specifies which trajectories are desirable so that the agent can identify optimal or near-optimal behaviors with respect to the objective. We want the agent to learn the optimal behavior for achieving the specified objective by interacting with the environment. At each step, the agent interacts with the environment by observing the current state of the system, taking an action based on its policy, and transitioning to the next state. Over repeated interactions, the agent learns a policy that maximizes the objective.

1.1 The Reward Objective

Suppose a practitioner would like to specify the objective for the robot "reach the goal and do not step on water." How should they communicate this objective to the agent and ensure that the agent learns the intended behavior to achieve the objective?

The prelevant way to specify an objective is via reward functions. A reward function specifies a scalar value, a reward, for each state of the system. The desired trajectories are

those with higher cumulative rewards.

Consider a robot agent in a gridworld environment to concretize this challenge, as shown in Figure 1.1. The agent starts in the upper left corner, and at every step, it can either attempt to move to a neighboring tile or stay in place. Each move of the agent is slippery — the agent has a 0.1 probability of slipping into a cardinal direction when trying to move to a neighboring tile. The blue tiles are water regions, and the flag is the goal.

1.1.1 Specification

In reinforcement learning, the standard practice to specify this objective is to encode it as a reward function [10]. A natural choice for the above example is to assign a positive reward $r_{\text{goal}} > 0$ once the agent reaches the goal and a negative reward $r_{\text{water}} < 0$ every time the agent steps on water. The objective the practitioner communicates to the agent is to maximize the cumulative discounted rewards over time for some discount factor $\gamma \in [0, 1)$ chosen by the practitioner. Then, they employ a reinforcement learning algorithm on the agent so that using this algorithm, the agent interacts with the environment and learns a good behavior that maximizes the cumulative discounted rewards.

The reward-function objective is well studied [10]. Due to its versatility, researchers have adopted the reward-function objective as the de facto standard of objective specification in reinforcement learning.

1.1.2 Guarantee

Another important aspect of reinforcement learning is obtaining a guarantee on the learned behavior. Specifically, we aim to specify an objective and let the agent learn a good policy. Thus, we need some assurance of how close to optimal the learned policy is.

The conventional reward-based reinforcement-learning objectives include infinite-horizon discounted and finite-horizon cumulative rewards. These objectives have a desirable property: There are reinforcement-learning algorithms that learn a near-optimal policy with high probability with a number of samples depending only on the parameters known by the algorithm [11]. We call these algorithms *probably approximately correct* (PAC), and these objectives *PAC-learnable* under reinforcement learning. PAC-learnability is essential: If an objective is not PAC-learnable, then the hope of ensuring learning a near-optimal policy in a finite amount of resources is lost, and the objective is effectively intractable to learn under reinforcement learning.

In the example, the reward function objective given by the cumulative discounted rewards is PAC-learnable. Therefore, the practioner can employ a reinforcement-learning algorithm

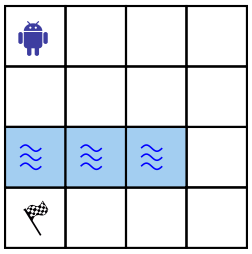


Figure 1.1: A robot in a gridworld. The robot starts in the upper left corner. The flag is the goal; the blue tiles are water region.

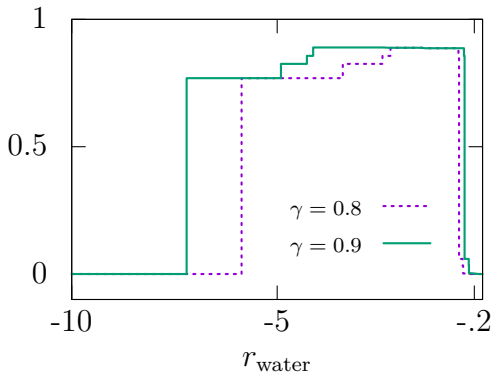


Figure 1.2: Probability of reaching the goal without stepping on water as a function of the reward r_{water}.

providing a PAC-learnability guarantee, such as RMax [12], and be assured that the agent will learn a near-optimal policy with high probability with a finite number of samples.

1.1.3 Reward Hacking

However, rewards are often imperfect surrogates to the true objective, causing the *reward hacking* phenomenon, where agents exploit rewards and not achieve the true objective.

In the example, for a fixed value of r_{goal}, if the reward r_{water} is not sufficiently negative, the agent might exploit the reward function by stepping on water to reach the goal faster. However, if the reward r_{water} is too negative, the agent might choose to stay in the starting white tile, just so that it never risks stepping on water — the agent finds out that the best way to avoid stepping on water is not to move at all. Figure 1.2 shows the exact probability of reaching the goal without stepping on water as a function of r_{water} in the example gridworld. We see that on both ends of the spectrum of r_{water}, the probability of satisfying the true objective is close to zero. Around the range of $r_{\text{water}} \in [-4, -0.5]$, that probability reaches the maximum. Even worse, the discount factor γ further complicates the matter. Figure 1.2 shows two different discount factors $\gamma = 0.8$ and $\gamma = 0.9$, and we see that the range of r_{water} that achieves the true objective varies with γ.

A good choice of rewards and discount factors depends on the environment dynamics, which is unknown in the reinforcement learning setting. Conventional reinforcement learning left the practitioner with the challenge of choosing the suitable rewards and the discount factor to ensure that the agent behaves optimally concerning the true objective. This challenge is insurmountable in practice due to the unknown environment dynamics, making agent behaviors unpredictable and suboptimal.

Motivation for a Framework for General Objectives Fundamentally, the root cause of reward hacking is because the reward objective, while a valid objective, is not the true objective "reach the goal and do not step on water." Instead, the reward objective is a surrogate to the true objective, meaning the practioner have chosen to use the reward objective as a proxy to the true objective, in the hope that if the agent behaves well with respect to the reward objective, then the agent also behaves well with respect to the true objective. However, this hope is not a guarantee, and the mismatch of the behaviors with respect to the reward objective and the true objective due to the surrogate nature of the reward objective leads to the problem of reward hacking.

If the practitioner could specify the true objective directly, and if the agent could learn the optimal behavior with a guarantee with respect to the true objective, then the agent behavior would be predictably optimal. From this reasoning, three key questions arise:

- How to formally specify the true objectives?
- Which objectives are learnable under the conventional PAC-learnability guarantee in reinforcement learning?
- What to do for general objectives that are not learnable under the conventional PAC-learnability guarantee?

1.2 Beyond Reward-based Objectives

Despite the adoption and the desirable PAC-learnability property of reward-based objectives, they have the key limitation of being a surrogate to the true objective, which leads to reward hacking. The root cause of this limitation is fundamentally due to the reward function's being a surrogate to the true objective, as I have discussed above, and a natural solution to address this limitation is to specify the true objective directly.

1.2.1 Example with Logical Specification

A way to specify the objective in our example is through logical specification.

1.2.1.1 Specification

Logical specifications define objectives using predicates on the environment, combined with logical operators such as conjunction ($\wedge$) and disjunction ($\vee$). These predicates express properties of the environment's state, and logical formulas express logical conditions that

must be met at a given step of the trajectory. For example, in a gridworld environment, predicates such as goal and water expresses whether an agent has reached the goal or stepped on water, respectively. The formula ¬water expresses that the agent should not step on water.

To model temporal aspects of the objective, we introduce modalities that describe how properties evolve over time. One such formalism is Linear Temporal Logic (LTL) [13] that incorporates temporal operators such as G (always), F (eventually) and U (until) to specify objectives over sequences of states. For example, in the gridworld environment, the formula $\mathsf{G}\,\neg$water expresses that the agent should never step on water, and the formula $\mathsf{F}\,$goal expresses that the agent should eventually reach the goal.

By combining predicates, logical operators, and temporal operators, LTL can express complex objectives. Our example objective "eventually reach the goal and never step on water" is expressible as the LTL formula $\mathsf{F}\,\text{goal} \wedge \mathsf{G}\,\neg\text{water}$, capturing both a future requirement (goal must be reached) and a safety constraint (water must never be visited).

LTL-based objectives have been widely studied in reinforcement learning [1, 2, 14–17]. Given an LTL formula as an objective, each *trajectory* of the system either satisfies or violates the formula. The goal of the optimal policy is to maximize the probability of satisfying the specified objective.

1.2.1.2 PAC-learnability

The general class of LTL objectives consists of *infinite-horizon objectives* that require inspecting infinitely many steps of a trajectory to determine if the trajectory satisfies the objective. For example, consider the objective $\mathsf{F}\,\mathit{goal}$ (eventually reach the goal). Given an infinite trajectory, the objective requires inspecting the entire trajectory in the worst case to determine whether it violates the objective.

Despite the above developments on reinforcement learning with LTL objectives, the infinite-horizon nature of these objectives presents challenges that have been alluded to—but not formally treated—in prior work. Henriques et al. [18], Ashok, Křetínský, and Weininger [19], and Jiang et al. [20] noted slow learning times for infinite-horizon properties. Littman et al. [21] provided a specific environment that illustrates the intractability of learning for a specific infinite-horizon objective, arguing for the use of a discounted variant of LTL.

To my knowledge, the learnability of LTL objectives had not been formally analyzed and understood until my work [22].

1.2.2 Other Specifications in the Literature

In recent years, researchers have introduced various objectives beyond reward-based objectives [1, 16, 21, 23–25].

1.2.2.1 Specification

Various works introduce the specification of a class objectives, for example:

- Littman et al. [21] introduced the Geometric Linear Temporal Logic (GLTL) objective, which extends LTL by associating each temporal operator with a geometrically distributed horizon. This objective defines tasks with probabilistic time bounds, enabling RL agents to optimize policies that satisfy temporal constraints within expected time frames while accounting for uncertainty in task durations.

- Camacho et al. [23] introduced the Reward Machine objective, which augments standard reward-based objectives by encoding the reward function as a finite state automaton. This objective enables history-dependent rewards, allowing agents to optimize behavior based on structured, temporally extended task specifications rather than instantaneous rewards.

- Bozkurt et al. [1] introduced the Limit-Deterministic Büchi Automaton (LDBA) objective as a surrogate to LTL objectives. It enforces history-dependent constraints on agent behavior using LDBAs. This objective incorporates history-dependent discount factors, rewards, and an augmented action space, ensuring that agents optimize long-term behaviors in accordance with LTL specifications.

- Hahn et al. [16] introduced the Omega-Regular objective, which generalizes LTL-based objectives to omega-regular properties. This objective ensures that RL agents optimize policies that satisfy complex, infinite-horizon temporal constraints by reducing the problem to an almost-sure reachability objective in limit-deterministic Büchi automata.

- Giacomo et al. [24] introduced the Finite-Trace Temporal Logic objective, which specifies RL objectives using Linear Temporal Logic over finite traces (LTLf) and Linear Dynamic Logic over finite traces (LDLf). These objectives enforce constraints on finite-length executions, enabling agents to optimize policies that adhere to safety, fairness, or other behavioral constraints over limited time horizons.

- Jothimurugan, Alur, and Bastani [25] introduced the Composable Specification objective, which provides a modular and reusable way to specify RL objectives. This objective enables the decomposition of complex control tasks into simpler, composable sub-objectives

that can be combined and reused across different tasks, improving learning efficiency and policy synthesis.

Researchers introduced formal specifications for these objectives. They introduced reinforcement-learning algorithms for these objectives and showed that they empirically learn well-behaving policies with finitely many samples and computational resources.

1.2.2.2 PAC-learnability

Despite the advances in empirical algorithms for these objectives, not all objectives are PAC-learnable: In particular, as this thesis will cover, my work [22] proved that infinite-horizon LTL objectives are not PAC-learnable. Therefore, to the end of having assurance on learning outcomes, I desire to understand the PAC-learnability of general objectives.

Some previous works [14, 18, 19, 26] address the PAC-learnability of particular objectives. However, these analyses give reinforcement-learning algorithms for particular objectives and do not generalize to others. Previous work [27] gave a framework of reductions between objectives whose flavor of generality is most similar to my work; however, they did not give a condition for when an objective is PAC-learnable. To my knowledge, the PAC-learnability of the objectives in Bozkurt et al. [1], Sadigh et al. [2], Hahn et al. [16], Hasanbeig et al. [17], Littman et al. [21], Camacho et al. [23], and Jothimurugan, Alur, and Bastani [25] are previously not known.

Relevant to model-based reinforcement learning, Bazille et al. [28] showed that it is impossible to learn the transitions of a Markov chain such that the learned and true models agree on all first-order behaviors. However, this result does not apply to the general reinforcement-learning setting.

1.2.3 Beyond the Literature

To my knowledge, prior to my work [29], research had predominantly focused on specific classes of objectives, such as LTL or reward machines introduced above, without addressing the challenge of defining general objectives beyond these structured formalisms. This unaddressed challenge leaves critical gaps in both specification and learning: Given an arbitrary objective, for instance, one specified in natural language, it is unclear how to formally specify that objective if it does not fit into any existing formalism in the literature. Moreover, it is unclear how to learn optimal behavior for that objective.

o1	Reach goal without stepping on water
o2	Do **o1** within $n = 15$ steps
o3	Do **o1** within $n \sim \text{Geom}(\frac{1}{15})$ steps
o4	Do **o1**, then retrace the steps back
o5	Do **o4** within $n = 30$ steps
o6	Repeat **o4** forever

Table 1.1: Example objectives in the gridworld Figure 1.1.

1.2.3.1 Specification

To illustrate this gap, consider an example objective in the gridworld "reach the goal without stepping on water and then retrace the steps back." This objective is not expressible as an LTL formula, as LTL does not have a mechanism to specify the retracing behavior.

More generally, Table 1.1 gives six example objectives in the gridworld. The objective **o1** is expressible as the LTL formula $\mathsf{F}\,\text{goal} \wedge \mathsf{G}\,\neg\text{water}$, as I have discussed above. The objectives **o2** and **o5** are also expressible as LTL formulas, albeit the LTL formulas have a combinatorial size concerning the number of steps n in the objectives. Prior work [25, 30] introduced specification formalisms that allow expressing objectives the objectives **o2** and **o5** in a more concise way. The other objectives are not expressible by known specification formalisms in the literature.

1.2.3.2 PAC-learnability

Since the formal specification of general objectives was not well understood before my work [29], the PAC-learnability of general objectives beyond reward-based and LTL objectives also remained open. In particular, I do not know which general objectives are learnable under the classic definition of PAC-learnability in reinforcement learning. I do not know what to do when general objectives are not classically PAC-learnable if such objectives exist.

1.3 Thesis: A Framework for General Objectives

To the end of addressing the questions and gaps as outlined above, I revisit the foundation of reinforcement learning objectives by building a framework for specifying and learning general reinforcement-learning objectives. I highlight the term "general" to mean all objectives that are functions of trajectories of the system. In particular, general objectives encompass all objectives in the literature, including reward-based, LTL, and other objectives satisfying the general mathematical definition. I investigate the specification and learnability problem of

objectives from first principles under this framework. In particular, this dissertation will address the following research questions:

- How to specify general reinforcement-learning objectives?
- Which general reinforcement-learning objectives are learnable under the classic definition of PAC-learnability in reinforcement learning?
- What to do when general reinforcement-learning objectives are not classically PAC-learnable? In particular, how to relax the definition of learnability so that general reinforcement-learning objectives become learnable in a relaxed sense?

To the end of addressing these questions, this thesis draws from my previous contributions [22, 29] and my ongoing investigation on learnability of non-computable objectives. The thesis naturally developes in three stages. In the first stage, I start with a well-studied, practically relevant class of objectives, namely the LTL objectives, and give my result the learnability of these objectives. In the second stage, I generalize the result to general objectives, and give a sufficient condition for learnability of any general objective. In the third stage, I investigate the learnability of non-computable objectives, and propose a relaxed definition of learnability suitable for non-computable objectives.

1.3.1 LTL Objectives

I leverage the probability correct in MDPs (PAC-MDP) framework [11, 31] to state and prove a theorem that reinforcement learning for LTL objective is PAC-learnable if and only if the objective is belongs to a class called *finitary*, the converse of infinite-horizon objectives. In particular this theorem implies that infinite-horizon LTL objectives are intractable, meaning no reinforcement-learning agents can identify a near-optimal behavior for infinite-horizon LTL objectives with confidence in finite time.

The intuition for this intractability is: Any finite number of interactions with an environment with unknown transition dynamics is insufficient to identify the environment dynamics perfectly. Moreover, for an infinite-horizon objective, a behavior's satisfaction probability under the inaccurate environment dynamics can differ arbitrarily from its satisfaction probability under the true dynamics. Consequently, a learner cannot guarantee with any confidence that it has identified near-optimal behavior for an infinite-horizon objective.

In my example, the objective **o1** is an infinite-horizon LTL objective, and suffers from this intractability. The objectives **o2** and **o5** are finitary LTL objectives, and PAC-learnable.

1.3.2 PAC-learnability of Computable Objectives

I address the question by giving sufficient conditions for PAC-learnability. Specifically, I analyze PAC-learnability in both the information-theoretic setting, which only considers sample complexity, and the computation-theoretic setting, which also considers computability. I prove that, in the information-theoretic setting (resp. computation-theoretic setting), an objective is PAC-learnable if it is uniformly continuous (resp. computable). These conditions simplify the process of proving objectives' PAC-learnability. In particular, my conditions avoid constraints on environments, policies, or reinforcement-learning algorithms, requiring reasoning only about the objective itself. I provide example applications of these conditions to three objectives in the literature whose PAC-learnability was previously unknown and prove that they are PAC-learnable.

1.3.3 Towards Learnability of Non-Computable Objectives

Lastly, I take the first steps toward policy learning for non-computable objectives by addressing two fundamental challenges: representation and learnability. To represent non-computable objectives, I establish a formal framework where they are expressed as nested limits of computable functions, ensuring a universal and constructive characterization. For learnability, I introduce a weaker yet meaningful PAC-learnability criterion that requires learning near-optimal policies for a sequence of PAC-learnable approximations. This criterion ensures that, in the limit, the near-optimal policy for the approximation also becomes near-optimal for the true non-computable objective. I further develop a sufficient condition that simplifies proving this weaker PAC-learnability and demonstrate its applicability through a range of examples.

1.3.4 Examples Objectives

Table 1.2 presents the example objectives that serve as a unifying thread throughout this thesis. At the beginning of each chapter—Chapters 4 to 6—these objectives illustrate and motivate key concepts and results. At the end of each chapter, they are revisited to analyze the implications of the findings. Each chapter makes incremental progress in understanding their learnability. By the end of Chapter 6, I provide a comprehensive account of these example objectives' learnability. Figure 1.3 summarizes this final result, capturing how different classes of objectives are situated in the learnability landscape. While I defer a full discussion to later chapters, this figure provides a high-level view of the key distinctions and theoretical challenges addressed in this thesis.

Example Gridworld Objectives	
o1	Reach goal without stepping on water
o2	Do **o1** within $n = 15$ steps
o3	Do **o1** within $n \sim \text{Geom}(\frac{1}{15})$ steps
o4	Do **o1**, then retrace the steps back
o5	Do **o4** within $n = 30$ steps
o6	Repeat **o4** forever
Classic Reinforcement-learning Objectives	
cp	Cart Pole: Pole always stays upright
mc	Mountain Car: Eventually reach the goal
pd	Pendulum: Eventually up and balance forever
tx	Taxi: Pick and drop passengers sequentially
Classic Reward-based Objectives [32]	
Σ^γ	Discounted cumulative rewards
Σ^H	Finite-horizon cumulative rewards
$\lim \frac{\Sigma}{N}$	Limit-average rewards
Objectives Beyond Rewards in the Literature	
RM	Simple Reward Machine [23]
Boz	Bozkurt's surrogate objective for LTL [1]
GLTL	Geometric LTL [21]

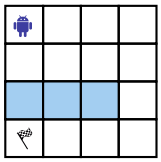


Table 1.2: Example objectives that serve as a guiding thread throughout the thesis.

1.4 Contributions

In this thesis, I present my contributions to reinforcement-learning objectives in three parts: LTL objectives, computable objectives, and non-computable objectives.

1.4.1 Linear Temporal Logic Objectives

I make the following contributions:

- A formalization of reinforcement learning with LTL objectives under the PAC-MDP framework [31, 33, 34], a standard framework for measuring sample complexity for reinforcement-learning algorithms; and a formal definition of LTL-PAC-learnable, a learnability criterion for LTL objectives.

- A statement and proof that: 1. Any infinite-horizon LTL formula is not LTL-PAC-learnable. 2. Any finite-horizon LTL formula is LTL-PAC-learnable. To that end, for any infinite-horizon formula, I give a construction of two special families of MDPs as counterexamples with which I prove that the formula is not LTL-PAC-learnable.

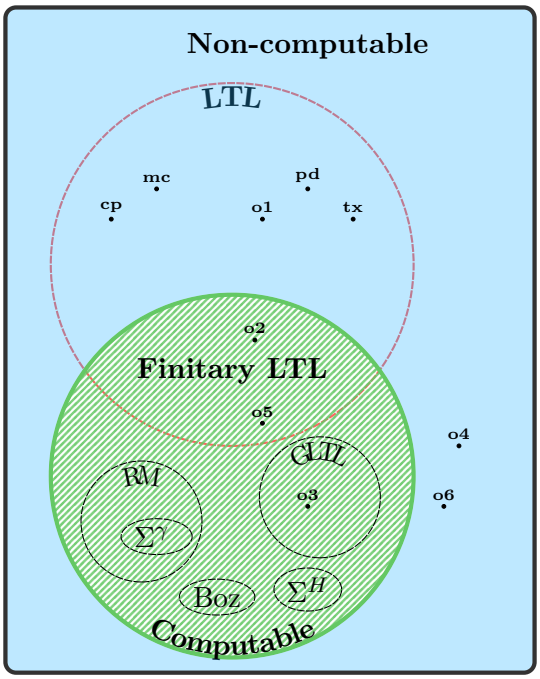


Figure 1.3: Preview of landscape of objectives' learnability by the end of the thesis.

- Experiments with current reinforcement-learning algorithms for LTL objectives that provide empirical support for my theoretical result.
- A categorization of approaches that focus on tractable objectives or weaken LTL-PAC-learnable guarantees and classification of previous approaches into these categories.

Implications for Relevant and Future Work My results provide a framework for categorizing approaches that either focus on tractable LTL objectives or weaken an algorithm's guarantees. As a result, I interpret various previous approaches as instantiations of the following categories:

- Work with finite-horizon LTL objectives, the complement of infinite-horizon objectives, to obtain guarantees on the learned behavior [18]. These objectives, like $a \wedge \mathsf{X}a$ (a is true for two steps), are decidable within a known finite number of steps.
- Seek a best-effort confidence interval [19]. Specifically, the interval can be trivial in the worst case, denoting that learned behavior is a maximally poor approximation of the optimal behavior.
- Make additional assumptions about the environment to obtain guarantees on the learned behavior [14, 35].

- Change the problem by working with LTL-like objectives such as: 1. relaxed LTL objectives that become exactly LTL in the (unreachable) limit [1, 2, 16, 17] and 2. objectives that use temporal operators but employ a different semantics [15, 21, 23, 24]. The learnability of these objectives is a potential future research direction.

1.4.2 Computable Objectives

I make the following contributions about reinforcement-learning objectives:

- In the information-theoretic setting, I prove that any uniformly continuous objective is PAC-learnable.
- In the computation-theoretic setting, I prove that any computable objective is PAC-learnable.
- I apply the above theorem to three objectives [1, 21, 23] from the literature, whose PAC-learnability was previously unknown, and show that they are PAC-learnable.

My result makes checking the PAC-learnability of existing objectives easier and guides the design of new PAC-learnable objectives.

1.4.3 Non-computable Objectives

I make the following contributions about non-computable objectives:

- I introduce a universal representation for non-computable objectives. In particular, any non-computable objective is representable by a nested limit of a computable objective.
- I introduce limit PAC-learnability, a relaxed definition of PAC-learnability, that is suitable for non-computable objectives. In particular, a non-computable objective is limit PAC-learnable if there is a sequence of PAC-learnable objectives whose optimal value converges to the optimal value of the non-computable objective.
- I give a condition for when a non-computable objective is limit PAC-learnable. The condition decomposes limit PAC-learnability to ensure a computable convergence rate of each layer in the nesting of the nested limit representation non-computable objective.
- I give various examples of non-computable objectives and use the given condition to show they are limit PAC-learnable.

1.5 Thesis Organization

The thesis is organized as follows:

In Chapter 2, I revisit the preliminaries relevant to this thesis, including reward-based reinforcement learning and the PAC-MDP framework.

In Chapter 3, I introduce the framework for expressing general reinforcement-learning objectives beyond rewards. I then present the generalization of the PAC-MDP framework to accommodate these general objectives. At the end of this chapter, I revisit the example objectives from Table 1.2 to illustrate the framework's applicability.

In Chapters 4 to 6, I present the core contributions of this thesis:

- In Chapter 4, I present my results on the learnability of LTL objectives.
- In Chapter 5, I present my results on computable objectives.
- In Chapter 6, I present my results on non-computable objectives.

At the end of each chapter in Chapters 4 to 6, I revisit the example objectives from Table 1.2 to contextualize the presented results of the chapter.

Finally, in Chapter 7, I discuss the broader implications of my contributions to AI research, suggest some future directions, and conclude the thesis.

Chapter 2

Preliminaries

The thesis uses tools from theoretical reinforcement learning and computability theory. This chapter provides a brief overview of basic notations and preliminary concepts in reinforcement learning with rewards to ground the discussion in the subsequent chapters.

2.1 Basic Notations

Unless otherwise specified, I use the following notations throughout the thesis.

For usual mathematical objects, I use the following notations: naturals $\mathbb{N}$, integers $\mathbb{Z}$, rationals $\mathbb{Q}$, reals $\mathbb{R}$. I use $\mathbb{1}\{e\}$ to denote the indicator function that outputs 1 if the condition e is true and 0 otherwise.

Sequences over Finite Set Let Σ be a finite set. When it is a finite set of symbols, I call it an *alphabet*. I denote the set of all finite-length sequences over Σ as Σ^* and the set of all infinite-length sequences over Σ as Σ^ω. In formal logic and computability theory, researchers often call sequences over Σ *words* or *strings*. In reinforcement learning, when Σ represents states-action pairs, researchers often call them *trajectories*. I use the terms *word*, *stream* and *trajectory* interchangeably, choosing the one that best aligns with the concept under discussion. In the context of programming, I also call finite-length sequences *lists* and infinite-length sequences *streams*.

Topology and Metric Spaces A topology on a set X is a collection of subsets of X that satisfy that: the empty set and X are open, arbitrary unions of open sets are open, and finite intersections of open sets are open. A function $f\colon X \to Y$ is continuous if the preimage of every open set in Y is open in X.

A metric space is a tuple (X, d), where X is a set and d is a metric on X. The metric d satisfies the non-negativity, symmetry, and triangle inequality properties. A metric space is complete if every Cauchy sequence in X converges to a limit in X. A metric induces a topology on X, where a set $U \subseteq X$ is open if, for every $x \in U$, there exists an $\epsilon > 0$ such that the ϵ-ball around x is contained in U.

An open cover of a set X is a collection of open sets whose union contains X. A space is compact if every open cover has a finite subcover.

Probabilities and Stochastic Processes A probability space is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where: Ω is a sample space, $\mathcal{F}$ is a σ-algebra of events, and $\mathbb{P}$ is a probability measure.

In the context of this thesis, I often consider probability spaces over infinite-length words: $\Omega = \Sigma^\omega$. For such spaces, $(\Sigma^\omega, d_{\Sigma^\omega})$ is a complete metric space, where d_{Σ^ω} is the metric

$$d_{\Sigma^\omega}(w_1, w_2) = 2^{-L_{\text{prefix}}(w_1, w_2)}$$

and $L_{\text{prefix}}(w_1, w_2)$ is the length of the longest common prefix of w_1 and w_2. The σ-algebra $\mathcal{F}$ of Σ^ω is the Borel σ-algebra over Σ^ω, the smallest σ-algebra that contains all open sets in the topology induced by the metric d_{Σ^ω}.

2.2 Reinforcement Learning with Rewards

This section reviews reinforcement learning with reward-based objectives to establish a foundation for understanding conventional reinforcement learning settings. It serves as a precursor to the general objectives framework discussed in Chapter 3.

I start with an informal overview of Markov Processes, reward functions, and the standard reinforcement learning framework. Then, I introduce their formal treatments and connections to PAC-learnability in the context of reward-based objectives.

To illustrate these concepts, I use the girdworld example in Figure 1.2.

2.2.1 Gridworld Example

Recall the gridworld example from Chapter 1, where we send a robot agent into a gridworld environment. We intend to let the robot reach the goal without stepping on water. Figure 2.1 repeats the gridworld example for reference. A conventional reinforcement learning setup for this example includes the following components: An environment is modeled as a Markov Decision Process (MDP) with a reward function, a reward objective specifying how the rewards are accumulated over time, and a learning model that determines how the agent

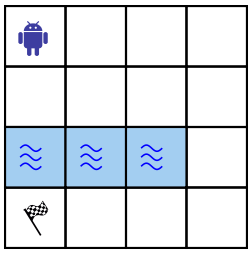


Figure 2.1: Gridworld example with a robot agent (blue) and water tiles (blue). The goal is to reach the flag (green) without stepping on water.

interacts with the environment. Then, we employ a reinforcement learning algorithm to learn a policy that maximizes the reward objective. We would prefer that the algorithm guarantees the learned policy's performance.

2.2.2 Markov Processes

Reinforcement learning traditionally assumes an autonomous agent interacts with an environment modeled as an MDP. An MDP consists of a set of states S, a set of actions A, and a transition probability function $P\colon (S \times A) \to \Delta(S)$, which maps state-action pairs to distributions over the next states.

Formally, a *Markov decision process with rewards* (MDP with rewards) is defined as a tuple $\mathcal{M} = (S, A, P, R, s_0)$, where:

- S is a finite set of states,
- A is a finite set of actions,
- $P\colon (S \times A) \to \Delta(S)$ is a transition probability function mapping state-action pairs to distributions over the next states,
- $R\colon S \to \mathbb{R}$ is a reward function, and
- $s_0 \in S$ is the initial state.

In the gridworld example, the robot's state is its coordinates on the grid, and the actions move up, down, left, or right. The initial state is— the upper left corner of the grid. The transition probability function models the robot's movement on the grid: The robot moves in the intended direction with high probability but may slip to a random neighboring cell.

The reward function $R\colon S \to \mathbb{R}$ assigns a numerical value to each state, quantifying the agent's immediate performance. For our example, as in Chapter 3, I specified the intended objective "reach the goal without stepping on water" using the reward function:

- A positive reward $r_{\text{goal}} > 0$ when the agent reaches the goal.
- A negative reward $r_{\text{water}} < 0$ when the agent steps on water.

A *policy* $\pi\colon ((S \times A)^* \times S) \to \Delta(A)$ maps state-action histories to a distribution over actions. A policy is *Markovian* if it depends only on the current state, $\pi\colon S \to \Delta(A)$. For the gridworld example, the Markovian policy specifies the robot's movement at each coordinate, while the non-Markovian policy specifies the robot's movement based on the current and previous coordinates.

An MDP and a policy induce a *discrete-time Markov chain* (Markov chain). A Markov chain is a tuple $\mathcal{D} = (S, P, s_0)$, where $P\colon S \to \Delta(S)$ defines the transition probabilities. The Markov chain induces a probability space over trajectories $w \in S^\omega$. In the gridworld example, the Markov chain models the behavior of the robot moving on the grid according to the policy. Intuitively, the robot's movement forms an infinite-length sequence of coordinates, a trajectory in the Markov chain, and the Markov chain captures the probability distribution over the possible trajectories.

2.2.3 Reward-Based Objectives

A reward-based objective assigns an accumulated value of the rewards mapped from the trajectory. This value reflects the agent's performance over time and guides the learning process. Three common reward-based objectives are cumulative discounted rewards, finite-horizon rewards, and infinite-horizon average reward.

Cumulative Discounted Rewards The *cumulative discounted rewards* objective is the γ-discounted infinite-horizon sum of rewards:

$$\kappa(w) \triangleq \sum_{i=0}^{\infty} \gamma^i \cdot R(w[i]),$$

where $\gamma \in [0, 1)$ is a discount factor, and $w[i]$ is the state at step i in the trajectory w. The *value* of the objective for the MDP $\mathcal{M}$ and a policy π starting from a state $s \in S$ is is the expectation of the objective under the probability space induced by the MDP and the policy starting from state s:

$$V^{\pi}_{\mathcal{M},\kappa}(s) = \mathsf{E}_{w \sim \mathcal{D}}[\kappa(w)]. \quad (\mathcal{D} \text{ induced by } \mathcal{M} \text{ with initial state replaced by } s \text{ and } \pi).$$

When the start state $s = s_0$, I drop the argument s and write $V^{\pi}_{\mathcal{M},\kappa}$ to denote the value of the objective under the MDP $\mathcal{M}$ and the policy π starting from the initial state s_0 of $\mathcal{M}$.

In Chapter 1, I used the cumulative discounted rewards objective to specify the robot's objective in the gridworld example. The robot aims to maximize the cumulative discounted rewards by reaching the goal while avoiding water. The discount factor γ controls the trade-off between immediate and future rewards: a higher γ emphasizes long-term rewards, while a lower γ emphasizes immediate rewards.

Finite-Horizon Rewards The *finite-horizon cumulative rewards* objective is the undiscounted sum up to a fixed horizon H:

$$\kappa(w) \triangleq \sum_{i=0}^{H} R(s_i).$$

The *value* of the objective for the MDP $\mathcal{M}$ and a policy π starting from a state $s \in S$ is is the expectation of the objective under the probability space induced by the MDP and the policy starting from the initial state s_0:

$$V^{\pi}_{\mathcal{M},\kappa} = \mathsf{E}_{w\sim\mathcal{D}}[\kappa(w)]. \quad (\mathcal{D} \text{ induced by } \mathcal{M} \text{ and } \pi).$$

Infinite-Horizon Average Reward Another reward-based objective is the *infinite-horizon average reward*. The standard formulation of average reward objective [32] is:

$$V_{\text{avg}}(\mathcal{D}) \triangleq \lim_{T\to\infty} \frac{1}{T} \mathsf{E}_{\mathcal{D}}\left[\sum_{i=0}^{T-1} R(w[i])\right]. \tag{2.1}$$

This formulation evaluates the steady-state behavior of a policy in the Markov chain $\mathcal{D}$ induced by the MDP and the policy. Unlike infinite-horizon discounted cumulative rewards, it balances rewards over an infinite time horizon rather than emphasizing near-term outcomes. Further, the average reward objective as defined by Equation (2.1) is a function over the Markov chain $\mathcal{D}$, rather than individual trajectories. The value of the average reward objective for the MDP $\mathcal{M}$ and a policy π is the average reward of the Markov chain $\mathcal{D}$ induced by the MDP and the policy starting from the initial state $s \in S$:

$$V^{\pi}_{\mathcal{M},\text{avg}}(s) = V_{\text{avg}}(\mathcal{D}) \quad (\mathcal{D} \text{ induced by } \mathcal{M} \text{ with initial state replaced by } s \text{ and } \pi).$$

When the start state $s = s_0$, I drop the argument s and write $V^{\pi}_{\mathcal{M},\text{avg}}$ to denote the value of the objective under the MDP $\mathcal{M}$ and the policy π starting from the initial state s_0 of $\mathcal{M}$.

2.2.4 Learning Models

I consider two learning models: planning with a generative model and reinforcement learning. In the planning setting, the agent has access to a generative model of the MDP, enabling it to sample the next states given any current state and an action. In the reinforcement learning setting, the agent cannot sample transitions from an arbitrary state but must follow the environment's transitions or reset to the initial state.

Our gridworld example applies to both learning models, and the choice is typically made based on the practical problem setup. In particular, as an example, if we are training the robot in a simulator that allows simulation from arbitrary states, we would choose to employ the planning-with-a-generative-model setting to learn an optimal policy by sampling transitions from the simulator. On the other hand, if the robot is a physical robot deployed in an unknown physical territory, the reinforcement learning setup must be used to learn the optimal policy through interaction.

Planning with a Generative Model In the planning-with-a-generative model setting, the agent accesses a *generative model* of the MDP. This model allows the agent to sample transitions by specifying a state and action. The agent uses these samples to learn a policy that maximizes the reward objective.

A planning algorithm $\mathcal{A}$ includes:

- A sampling strategy $\mathcal{A}^{\mathrm{S}}$ to determine which state-action pairs to sample next, and
- A learning algorithm $\mathcal{A}^{\mathrm{L}}$ to derive a policy from the sampled transitions.

Reinforcement Learning Reinforcement learning involves direct interaction with the environment to optimize the reward objective. The agent observes state transitions and selects actions to maximize cumulative rewards. I view a reinforcement-learning algorithm as a special kind of planning-with-generative-model algorithm, where the sampling strategy $\mathcal{A}^{\mathrm{S}}$ follows the environment's transitions.

There are two reinforcement learning settings: episodic and non-episodic. They are distinct in whether the agent can reset the environment to the initial state during learning. I focus on the episodic setup for this thesis and justify this choice in Chapter 3. I provide a brief overview of both settings here for completeness.

The agent can reset the environment to the initial state in an episodic setting. The reset can be voluntary by the agent: The agent has a special reset action that brings the agent to the initial state; Alternatively, the problem setup can force the reset, such as the fixed finite time horizon in the finite-horizon cumulative rewards setting. In this setting, the quality of

the policy is typically evaluated over episodes, and the value of the objective concerns only the initial state s_0 of the MDP,

In the non-episodic setup, the agent must follow the natural transitions of the environment without resets. Learning proceeds over an infinite time horizon, and the value of the objective is concerned with each state in the trajectory.

2.2.5 Guarantees for Reinforcement Learning Algorithms

Various kinds of guarantees exist for reinforcement learning algorithms for reward-based objectives, such as various flavors of probably approximately correct (PAC) in MDP (PAC-MDP) guarantees and regret-style guarantees. The PAC-MDP guarantees originate from the supervised learning theory [36] but are adapted to the planning-with-generative model and reinforcement-learning settings. Throughout this thesis, I consider the reinforcement learning settings and do not discuss the supervised learning problem. Therefore, throughout this thesis, when I mention PAC guarantees, I refer to the PAC-MDP guarantees, and there is no confusion with PAC guarantees in the supervised learning setting.

Three types of guarantees are common in the literature:

- Supervised-learning-style probably approximately correct in MDPs (PAC-MDP) guarantee ensures that the learning algorithm learns a policy that achieves near-optimal performance with high probability in a bounded number of samples.
- Mistake-style PAC-MDP guarantee ensures that the number of mistakes the learning algorithm makes is bounded.
- Regret-style guarantee ensures that the total error of learning algorithm is bounded.

Various other guarantees exist in the literature, such as uniform PAC-MDP guarantee [37] and policy certificate guarantee [38], but I do not discuss them here for brevity.

Supervised-Learning-Style PAC-MDP Guarantee In this thesis, I focus on the supervised-learning-style PAC-MDP guarantee and justify this choice in Chapter 3. I thus review the supervised-learning-style PAC-MDP guarantee for reward-based objectives in the following. Given an MDP $\mathcal{M} = (S, A, P, R, s_0)$, a learning algorithm $\mathcal{A}$ is *PAC-MDP* (supervised-learning-style) if, for any $\epsilon > 0$, $\delta > 0$, and $0 \leq \gamma < 1$, it produces a policy π such that:

$$\mathsf{P}_{\mathsf{T} \sim \langle \mathcal{M}, \mathcal{A}^{\mathrm{S}} \rangle_N} \left(V_{\mathcal{M},\kappa}^{\mathcal{A}^{\mathrm{L}}(T)} \geq V_{\mathcal{M},\kappa}^{*} - \epsilon \right) \geq 1 - \delta.$$

The literature is also concerned with the learning algorithm's efficiency so that the learning algorithm's sample and computational complexity are polynomial in a set of relevant

parameters. Such an efficiency concern is objective dependent. The relevant parameters for discounted cumulative rewards are the number of states $|S|$, the number of actions $|A|$, the accuracy parameters $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$, the range of rewards $R_{\max} - R_{\min}$, and the effective horizon $\frac{1}{1-\gamma}$. The relevant parameters for finite-horizon cumulative rewards are the same, except it uses the horizon H instead of the effective horizon $\frac{1}{1-\gamma}$. There is no PAC guarantee for average reward without additional assumptions on the MDP [33].

For the gridworld example, the supervised-learning-style PAC-MDP guarantee ensures that the robot learns a policy that maximizes the cumulative discounted rewards with high probability in a bounded number of samples. However, note that this guarantee only holds for the cumulative discounted rewards objective, which is a surrogate for the true objective of reaching the goal without stepping on water. The guarantee does not directly ensure the robot's performance on the true objective.

In the next chapter, I introduce the general objectives framework, extending reward-based objectives to a broader class of objectives. I discuss the PAC-MDP guarantees for general objectives in Chapter 3. The general objectives framework generalizes the reward-based objectives and encompasses them as a special case.

Chapter 3

Foundation for General Reinforcement-learning Objectives

This section overviews the foundation of general reinforcement-learning objectives laid out by my previous contributions [22, 29]. I first informally overview background on Markov processes, the definition of general objectives, learning models, and objectives' PAC-learnability. I then give formal treatments of these concepts.

3.1 Overview

Conventional reward-based objectives are history-independent and Markovian: the reward at each step depends only on the current state and action. In contrast, general objectives are functions mapping from infinite-length trajectories to real numbers ranking the trajectories. To account for the distinction, I extend the framework of conventional reinforcement learning to accommodate general objectives. The following section gives an example general objective: the simple reward machine objective [23]. It serves as a first example of a general objective beyond the conventional reward-based objectives. More examples will follow throughout the thesis's progression.

Example: The Simple Reward Machine Specification Simple reward machines generalize from classic Markovian rewards to non-Markovian rewards. In particular, a simple reward machine is a kind of deterministic finite automaton. Each automaton transition has a reward value and a tuple of truth values for a set of propositions about the environment's state. The simple reward machine starts from an initial state. As the agent steps through the environment, a labeling function classifies the environment's current state to a tuple of truth values of a set of propositions. The simple reward machine then transits to the next

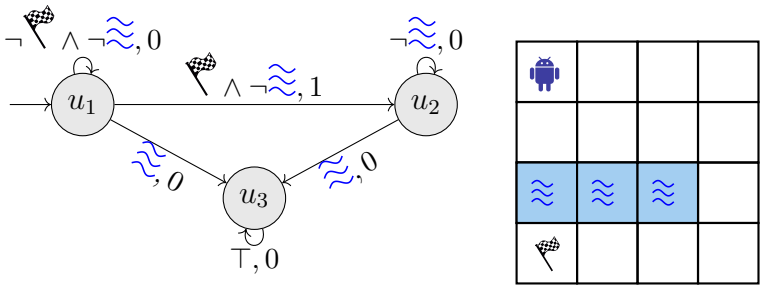


Figure 3.1: Left: simple reward machine. Right: environment.

states according to the tuple. During each transition, the agent collects a scalar reward along the transition of the simple reward machine. The overall objective is to maximize the γ-discounted sum of collected rewards. The formal definition of a simple reward machine given by Camacho et al. [23] is:

Definition 3.1.1 (Simple Reward Machine)**.** *Given a finite set* Π *called the propositions, a simple reward machine over* Π *is a tuple* $(U, \delta_u, \delta_r, u_0, \gamma)$*, where* U *is a finite set of states,* $\delta_u \colon (U \times 2^{\Pi}) \to U$ *is a deterministic state transition function,* $\delta_r \colon (U \times U) \to \mathbb{Q}$ *is a deterministic reward function,* u_0 *is an initial state, and* $\gamma \in \mathbb{Q}$ *is a discount factor.*

Figure 3.1 shows an example simple reward machine and an accompanying grid environment. The states of the simple reward machine are $\{u_1, u_2, u_3\}$. The labeling function for this particular environment maps grid locations $(1, 0)$ and $(2, 0)$ to "water" (≋) and $(3, 0)$ to "goal" (⚑). Each transition of the simple reward machine is labeled by a tuple of truth values of the two propositions ("water" and "goal"). For example, u_1 transits to u_2 and produces a reward of 1 if the environment's current state is labeled as "goal" but not "water".

The framework for general objectives closely mirrors the conventional framework for reward-based objectives reviewed in Chapter 2.

Markov Processes The definition involving MDPs and policies remains the same as the conventional reward-based objectives, except that I consider MDPs without an explicit reward function. In particular, I consider an autonomous agent situated in an environment with unknown dynamics modeled as an MDP, defined by a set of states, actions, and a transition probability function that determines the probability distribution of moving from one state to another based on an action. It starts from an initial state and, under a policy — which can be either Markovian, using only the current state information, or non-Markovian, using a history of states and actions — gives rise to a discrete-time Markov chain (Markov chain). This chain further induces a probability space over trajectories of states.

In our example, the environment is the gridworld shown in Figure 3.1.

Objectives A general reinforcement-learning objective is a function that assigns values to the samples paths of the induced Markov chain, with higher values indicating more preferable outcomes. On the high level, I would like the agent to learn a policy that maximizes the expected value of the objective over the sample paths induced by the environment MDP and the policy. An objective can be specific to a particular environment or generic, applicable across various MDPs.

In our example, a simple reward machine R specifies an environment-generic objective $[\![R]\!]\colon \left(2^{\Pi}\right)^{\omega} \to \mathbb{R}$ given by:

$$[\![R]\!](w) = \sum_{k=1}^{\infty} \gamma^k \delta_r(u_k, u_{k+1}), \quad \forall k \geq 0.\, u_{k+1} = \delta_u(u_k, w[k]).$$

The set of features F corresponding to this environment-generic objective is the possible truth values of the propositions Π, that is $F = 2^{\Pi}$. The labeling function classifies each environment's current state and action to these features.

Learning Models The learning models are the same as in the conventional reward-based objectives. In particular, I consider two learning models: planning with a generative model and reinforcement learning.

Learnability of Objectives Our goal in specifying an objective is to learn a policy that performs well with respect to the objective. More importantly, to make the notion of "good performance" precise, We want to ensure that the learned policy is near-optimal and highly probable. The PAC-MDP framework formalizes this notion. To that end, I generalize the PAC-MDP framework from reward-based objectives to general objectives.

In a nutshell, a general objective is PAC-learnable if an algorithm, with high probability, learns a near-optimal policy for the objective in a finite amount of resources.

3.2 Markov Processes

A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, A, P, s_0)$, where S and A are finite sets of states and actions, $P\colon (S \times A) \to \Delta(S)$ is a transition probability function that maps a current state and an action to a distribution over next states, and $s_0 \in S$ is an initial state. The MDP is sometimes referred to as the *environment MDP* to distinguish it from any specific objective.

A *policy* for an MDP is a function $\pi\colon ((S \times A)^* \times S) \to \Delta(A)$ that maps a history of

states and actions to a distribution over actions.[1] If the policy only depends on the current state, it is called a *Markovian policy* and can be written as $\pi\colon S \to \Delta(A)$.

An MDP and a policy induce a Markov chain. A Markov chain is a tuple $\mathcal{D} = (S, P, s_0)$, where S is the set of states, $P\colon S \to \Delta(S)$ is a transition-probability function mapping states to distributions over next states, and $s_0 \in S$ is an initial state. The Markov chain induces a probability space over the infinite-length sequences $w \in S^\omega$.

3.3 Objectives

An objective assigns a value to each infinite-length trajectory of the system. The value designates the preferredness of each trajectory: the larger the value, the better the trajectory. The goal of an agent is to learn a policy that maximizes the expected value of the objective over the random trajectories induced by the system.

3.3.1 Environment-specific Objective

An *environment-specific objective* for an MDP $\mathcal{M} = (S, A, P, s_0)$ is a measurable function $\kappa\colon S^\omega \to \mathbb{R}$ on the probability space of the Markov chain $\mathcal{D}$ induced by $\mathcal{M}$ and a policy π. I say such an objective is environment-specific since it is associated with MDPs with a fixed set of states and actions.

The *value* of the objective for the MDP $\mathcal{M}$ and a policy π is the expectation of the objective under that probability space:

$$V^\pi_{\mathcal{M},\kappa} = \mathsf{E}_{w\sim\mathcal{D}}[\kappa(w)] \quad (\mathcal{D} \text{ induced by } \mathcal{M} \text{ and } \pi).$$

For example, the cumulative discounted rewards objective [32] with discount γ and a reward function $R\colon S \to \mathbb{R}$ is:

$$\kappa^{\text{reward}}(w) \triangleq \sum_{i=0}^{\infty} \gamma^i \cdot R(w[i]). \tag{3.1}$$

I consider only bounded objectives to ensure that the expectation exists and is finite.

The *optimal value* is the supremum of the values achievable by all policies: $V^*_{\mathcal{M},\kappa} = \sup_\pi V^\pi_{\mathcal{M},\kappa}$. A policy π is ϵ-optimal if its value is ϵ-close to the optimal value: $V^\pi_{\mathcal{M},\kappa} \geq V^*_{\mathcal{M},\kappa} - \epsilon$.

[1] X^* denotes all finite-length sequences of the elements of X. X^ω denotes all infinite-length sequences of the elements of X.

3.3.2 Environment-generic Objective

An objective defined above is environment-specific because it is associated with a fixed set of states and actions of the environment. However, sometimes, I would also like to talk about objectives in a form decoupled from any MDP. Examples that justify the usefulness include:

- The classical discounted cumulative rewards objective is not bound to any particular reward function. In particular, it makes sense to talk about the discounted sum in the objective (Equation (3.1)) with an abstracted reward function R.
- The objective of "reaching the goal state" in a gridworld environment is not bound to the size of the grid or the allowed actions.

Such decoupling is desirable as it allows specifying objectives independent of environments.

To that end, I define *environment-generic* objectives. The idea of such objectives is that a *labeling function* interfaces between the environment and the environment-generic objective. The definition decouples an environment-generic objective from environments by requiring different labeling functions for different environments.

Formally, an *environment-generic objective* is a measurable function: $\xi\colon F^\omega \to \mathbb{R}$, where F is a set called *features*. A *labeling function* maps the MDP's (current) states and actions to the features: $\mathcal{L}\colon (S \times A) \to F$. Composing ξ and the element-wise application of $\mathcal{L}$ induces an environment-specific objective. For example, the discounted cumulative rewards objective $\xi\colon \mathbb{Q}^\omega \to \mathbb{R}$ is $\xi(w) = \sum_{i=0}^{\infty} \gamma^i \cdot w[i]$. For each MDP, the labeling function is a classical reward function $\mathcal{L}\colon (S \times A) \to \mathbb{Q}$. [2]

The value of ξ for an MDP $\mathcal{M}$, a policy π, and a labeling function $\mathcal{L}$ is the value of the environment-specific objective induced by ξ, $\mathcal{M}$, π, and $\mathcal{L}$.

3.4 Planning with a Generative Model

A planning-with-generative-model algorithm [39, 40] has access to a *generative model*, a sampler, of an MDP's transitions but does not have direct access to the underlying probability values. It can take any state and action and sample a next state. It learns a policy from those sampled transitions.

Formally, a planning-with-generative-model algorithm $\mathcal{A}$ is a tuple $(\mathcal{A}^{\mathrm{S}}, \mathcal{A}^{\mathrm{L}})$, where $\mathcal{A}^{\mathrm{S}}$ is a *sampling algorithm* that drives how the environment is sampled, and $\mathcal{A}^{\mathrm{L}}$ is a *learning algorithm* that learns a policy from the samples obtained by applying the sampling algorithm.

[2] For simplicity, I let objective specifications use rationals instead of reals so that they admit a finite representation. Nonetheless, my analyses also generalize to objective specifications that contain reals.

In particular, the sampling algorithm $\mathcal{A}^{\mathrm{S}}$ is a function that maps from a history of sampled environment transitions $((s_0, a_0, s'_0) \dots (s_k, a_k, s'_k))$ to the next state and action to sample (s_{k+1}, a_{k+1}) , resulting in $\mathsf{s}'_{k+1} \sim \mathsf{P}(\,\cdot \mid s_{k+1}, a_{k+1}\,)$. Iterative application of the sampling algorithm $\mathcal{A}^{\mathrm{S}}$ produces a sequence of sampled environment transitions.

The learning algorithm is a function that maps that sequence of sampled environment transitions to a non-Markovian policy of the environment MDP. Note that the sampling algorithm can internally consider alternative policies as part of its decision of what to sample. Also, note that I deliberately consider non-Markovian policies since the optimal policy for a general objective is non-Markovian in general.

3.5 Reinforcement Learning

In reinforcement learning, an agent is situated in an environment MDP and only observes state transitions. We also allow the agent to reset to the initial state as in Fiechter [31].

Some works on reward-based objectives assume a non-episodic setting, where resetting the environment is not required [33, 41]. This assumption is feasible for the case of reward-based objectives because the value of discounted or undiscounted reward objectives are definable starting from any point in the trajectory.

In contrast, this thesis focuses on the episodic setting. This choice is motivated by the fact that the value of a general objective is only well-defined as the expected value of the objective function over trajectories originating from the initial state. Consequently, the agent must gather sufficient knowledge about the environment's dynamics from the initial state to optimize the objective. Therefore, a reset mechanism that allows the agent to repeatedly begin from the initial state is a necessary requirement for addressing the general objectives considered in this thesis.

Same as in the reward-based objectives case, I view a reinforcement-learning algorithm as a special kind of planning-with-generative-model algorithm $(\mathcal{A}^{\mathrm{S}}, \mathcal{A}^{\mathrm{L}})$ such that the sampling algorithm always either follows the next state sampled from the environment or resets to the initial state of the environment. and only has the choice for the next action.

3.6 Probably Approximately Correct in MDPs

A successful planning-with-generative-model (or reinforcement-learning) algorithm should learn from the sampled environment transitions and produce an optimal policy for the objective in the environment MDP. However, since the environment transitions may be stochastic, I cannot expect an algorithm to always produce the optimal policy. Instead, I seek an

algorithm that, with high probability, produces a nearly optimal policy. The PAC-MDP framework [31, 33, 34], which takes inspiration from probably approximately correct (PAC) learning [36], formalizes this notion. The PAC-MDP framework requires efficiency in both sampling and algorithmic complexity. In this thesis, I only consider sample efficiency and thus omit the requirement on algorithmic complexity. Next, I generalize the PAC-MDP framework from reinforcement-learning with a reward objective to planning-with-generative-model with a generic objective.

Definition 3.6.1. *Given an objective κ, a planning-with-generative-model algorithm $(\mathcal{A}^S, \mathcal{A}^L)$ is* κ-PAC*(probably approximately correct for objective κ) in an environment MDP $\mathcal{M}$ if, with the sequence of transitions T of length N sampled using the sampling algorithm $\mathcal{A}^S$, the learning algorithm $\mathcal{A}^L$ outputs a non-Markovian ϵ-optimal policy with probability at least $1-\delta$ for any given $\epsilon > 0$ and $0 < \delta < 1$. That is:*

$$\mathsf{P}_{\mathsf{T}\sim\langle\mathcal{M},\mathcal{A}^S\rangle_N}\left(V_{\mathcal{M},\kappa}^{\mathcal{A}^L(T)} \geq V_{\mathcal{M},\kappa}^{\pi^*} - \epsilon\right) \geq 1-\delta.$$

We use $\mathsf{T}\sim\left\langle\mathcal{M},\mathcal{A}^{\mathrm{S}}\right\rangle_N$ to denote that the probability space is over the set of length-N transition sequences sampled from the environment $\mathcal{M}$ using the sampling algorithm $\mathcal{A}^{\mathrm{S}}$. For brevity, I will drop $\left\langle\mathcal{M},\mathcal{A}^{\mathrm{S}}\right\rangle_N$ when it is clear from context and simply write $\mathsf{P}_{\mathsf{T}}(.)$ to denote that the probability space is over the sampled transitions. If N is finite with respect to the parameters ϵ, δ, $|S|$, and $|A|$ and the objective κ, then we say the algorithm is κ-PAC. The algorithms is further *sample efficient* if the dependence is polynomial:

Definition 3.6.2. *Given an objective κ, a κ-PAC planning-with-generative-model algorithm is* sample efficiently κ-PAC *if the number of sampled transitions N is asymptotically polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $|S|$, $|A|$.*

Note that I allow the polynomial to have constant coefficients that depends on κ.

3.6.1 Learnability of Objectives

A good reinforcement-learning algorithm should provide some form of guarantee that it learns the good policy that maximizes the given objective. In particular, I let the algorithm seek a near-optimal policy with high probability.

Definition 3.6.3 (PAC Algorithm for Environment-specific Objective)**.** *Given an objective κ, a reinforcement-learning algorithm $(\mathcal{A}^S, \mathcal{A}^L)$ is κ-PAC (probably approximately correct for objective κ) in an environment MDP $\mathcal{M}$ with N samples if, with the sequence of transitions T of length N sampled using the sampling algorithm $\mathcal{A}^S$, the learning algorithm $\mathcal{A}^L$ outputs*

an ϵ-optimal policy with probability at least $1-\delta$ for any given $\epsilon > 0$ and $0 < \delta < 1$. That is:

$$\mathsf{P}_{\mathsf{T}\sim\langle\mathcal{M},\mathcal{A}^S\rangle_N}\left(V_{\mathcal{M},\kappa}^{\mathcal{A}^L(T)} \geq V_{\mathcal{M},\kappa}^{*} - \epsilon\right) \geq 1-\delta.$$

We use $\mathsf{T}\sim\left\langle\mathcal{M},\mathcal{A}^{\mathrm{S}}\right\rangle_N$ to denote that the probability space is over the set of length-N transition sequences sampled from the environment $\mathcal{M}$ using the sampling algorithm $\mathcal{A}^{\mathrm{S}}$. We will simply write $\mathsf{P}_{\mathsf{T}}(.)$ when it is clear from the context.

We will consider two settings: the information-theoretic setting that considers only sample complexity and the computation-theoretic setting that considers computability.

Definition 3.6.4 (PAC-learnable Environment-specific Objective)**.** *In the information-theoretic setting (resp. computation-theoretic setting), an environment-specific objective κ is* κ-PAC-learnable *if there exists a function $C\colon (\mathbb{R}\times\mathbb{R}\times\mathbb{N}\times\mathbb{N}) \to \mathbb{N}$ such that, for all consistent environment MDPs for κ (i.e., the domain of κ uses the same set of states and actions as the MDP), there exists a κ-PAC reinforcement-learning algorithm with less than $C(\frac{1}{\epsilon},\frac{1}{\delta},|S|,|A|)$ samples (resp. computation steps).*

My definition focuses on the core tractability issue. Failure to respect my definition implies that PAC-learning is not achievable with finitely many samples (in the information-theoretic setting) or not computable (in the computation-theoretic setting). To that end, I have set the parameters of C to be the only quantities available to an algorithm under the standard assumptions of reinforcement learning. Specifically, since the transition dynamics are unknown, they are not parameters of C. Moreover, while some variants of PAC-learnability require C to be a polynomial to capture the notion of learning efficiency, I have dropped this requirement to focus on the core tractability issue.

We also define the PAC-learnability of environment-generic objectives, for both information-theoretic and computation-theoretic settings:

Definition 3.6.5 (PAC-learnable Environment-generic Objective)**.** *An environment-generic objective ξ is* ξ-PAC-learnable *if for all labeling functions $\mathcal{L}$, the objective κ induced by ξ and $\mathcal{L}$ is κ-PAC-learnable.*

Note that in the information-theoretic setting, I assume that the objectives κ and ξ are given as oracles: they take infinite-length inputs and return infinite-precision output with no computation overhead.

3.6.2 Established PAC-Learnable Objectives

The standard discounted cumulative rewards objective $\sum_{i=0}^{\infty}\gamma^i w[i]$ and the finite-horizon cumulative rewards objective $\sum_{i=0}^{H} w[i]$ are known to be PAC-learnable. The folklore intuition

is that these objectives "effectively terminate" in an expected finite-length horizon, and rewards farther out of the horizon diminish quickly. Later in Section 5.3, I formalizes this intuition by connecting it to the standard definition of the objective function's uniform continuity and computability. In particular, I will prove that uniformly continuous and computable objectives are PAC-learnable.

Chapter 4

Linear Temporal Logic Objectives

In Section 1.1.3, I discussed how reward objectives are surrogates to the true objective, and the mismatch between the two leads to the undesirable reward hacking problem. Recall our example objective in the gridworld "eventually reach the goal while avoiding water", Figure 1.2 shows the reward hacking problem in this example — a practitioner cannot know how to assign rewards and the discount factor so that the agent behaves according to the true objective, without knowing the environment dynamics.

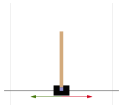
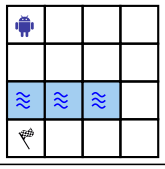
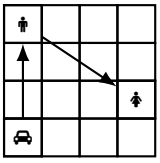
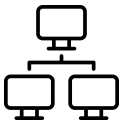
4.1 Overview

As a remedy to the reward hacking problem, researchers have sought to use logical objectives to specify the true objective, and proposed algorithms that attempt to learn behaviors that maximize the probability of satisfaction with the logical objectives. One such class of objectives is linear temporal logic (LTL) objectives.

4.1.1 Linear Temporal Logic Objective

LTL is a formal logic used initially to specify behaviors for system verification [13]. An LTL formula is built from a set of propositions about the state of the environment, logical connectives, and temporal operators such as G (always) and F (eventually). Many reinforcement-learning tasks are naturally expressible with LTL [21]. For some classic control examples, we can express: 1) cart-pole as $\mathsf{G}\,\mathit{up}$ (i.e., the pole always stays up), 2) mountain-car as $\mathsf{F}\,\mathit{goal}$ (i.e., the car eventually reaches the goal), and 3) pendulum-swing-up as $\mathsf{F}\,\mathsf{G}\,\mathit{up}$ (i.e., the pendulum eventually always stays up). Table 4.1 shows a collection of examples of LTL objectives in various tasks.

Table 4.1: Example LTL objectives in various tasks.

Name	Illustration	Description	LTL Formula
Assembly		In a time-critical manufacturing process, a robotic arm must drop an item at an exact timestep (e.g., the second step).	XXdrop
Cart-pole		The cartpole system must maintain balance by ensuring the pole always remains upright.	G up
Mountain-car		The car eventually reach the top of the hill.	F goal
Pendulum		The pendulum should eventually stabilize in an upright position and remain there indefinitely.	$\mathsf{F}\,\mathsf{G}$ up
System Monitoring		The software monitor must perform health checks infinitely often.	$\mathsf{G}\,\mathsf{F}$ check
Gridworld		Example objective in Chapter 1: Reach goal while avoiding water	F goal $\wedge$ $\mathsf{G}\,\neg$water
Taxi		The taxi must first pick up passenger one, then pick up passenger two, while ensuring sufficient fuel throughout.	$\mathsf{F}\,(\text{pickup}_1 \wedge \mathsf{F}\,\text{pickup}_2) \wedge \mathsf{G}$ fuel
Scheduler Fairness		The scheduler must schedule processes fairly, in the sense that if a process is enabled indefinitely, it must also be scheduled infinitely often. In other words, no single process shall take precedence indefinitely.	$\mathsf{G}\,(\mathsf{F}$ enabled $\Rightarrow$ F scheduled$)$

In our example, the true objective is expressible as the LTL formula $\mathsf{F}\,\text{goal} \wedge \mathsf{G}\,\neg\text{water}$. If the agent learns a behavior that maximizes the probability of satisfying this formula, then the agent will eventually reach the goal without stepping on water. Therefore, if the objective is learnable, meaning we could obtain a guarantee on the agent learning such behavior in a finite amount of interactions with the environment and computation, then we have avoided the reward hacking problem.

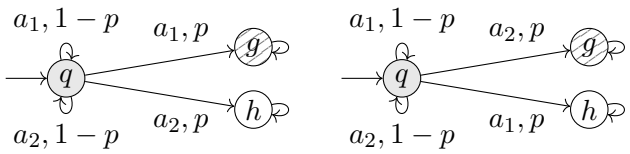


Figure 4.1: Two MDPs parameterized by p in range $0 < p < 1$. Action a_1 in the MDP on the left and action a_2 in the MDP on the right have probability p of transitioning to the state g. Conversely, action a_2 in the MDP on the left and action a_1 in the MDP on the right have probability p of transitioning to the state h. Both actions in both MDPs have probability $1 - p$ to loop around the start state s.

Researchers have thus used LTL as an alternative objective specification for reinforcement learning [1, 2, 14–17]. Given an LTL objective specified by an LTL formula, each trajectory of the system either satisfies or violates that formula. The agent should learn the behavior that maximizes the probability of satisfying that formula.

This chapter inspects the learnability of LTL objectives — whether we can obtain such a guarantee. I show that such a guarantee is possible only for a subset of LTL objectives called *finitary* and impossible for all other infinite-horizon LTL objectives. The main chapter will make this subset precise in this chapter and prove the above claim. But first, I illustrate this distinction of learnability between the two classes of LTL objectives through our gridworld example in Section 1.3.4

For our gridworld examples, the objective **o1** "eventually reach the goal without stepping on water" is not finitary and thus not learnable. The objective **o2** "eventually reach the goal and step on water in at most 15 steps" is finitary and thus learnable. In the following sections, I will use these two examples to illustrate the learnability of LTL objectives.

4.1.2 Infinite-horizon LTL Objective Example

The general class of LTL objectives consists of *infinite-horizon objectives* — objectives that require inspecting infinitely many steps of a trajectory to determine if the trajectory satisfies the objective. For example, consider a simplification of the objective **o1**: "eventually reach the goal", where I supposed no water exists in the grid world, and therefore dropped the requirement on not stepping on the water since it is trivially satisfied. Given an infinite trajectory, the objective requires inspecting the entire infinite trajectory in the worst case to determine whether it violates the objective since it may never reach the goal. Therefore, the objective is infinite-horizon. The following example illustrates the intractability of learning infinite-horizon objectives.

Suppose I send an agent into one of the MDPs in Figure 4.1, and want its behavior to

satisfy the objective "eventually reach the goal", where g is the goal state. This objective is expressible as the LTL formula $\mathsf{F}\,\text{goal}$. The optimal behavior is always to choose the action along the transition $q \to g$ for both MDPs (i.e., a_1 for the MDP on the left and a_2 for the MDP on the right). This optimal behavior satisfies the objective with probability one. However, the agent does not know which of the two MDPs it is in. The agent must follow its sampling algorithm to explore the MDP's dynamics and use its learning algorithm to learn this optimal behavior.

If the agent observes neither transition going out of q (i.e., $q \to g$ or $q \to h$) during sampling, it will not be able to distinguish between the two actions. The best it can do is a 50% chance guess and cannot provide any non-trivial guarantee on the probability of learning the optimal action.

On the other hand, if the agent observes one of the transitions going out of q, it will be able to determine which action leads to state h, thereby learning always to take that action. However, the probability of observing any such transition with N interactions is at most $1-(1-p)^N$. This is problematic: with any finite N, a value of p always exists such that this probability is arbitrarily close to 0. In other words, with any finite number of interactions, without knowing the value of p, the agent cannot guarantee (a non-zero lower bound on) its chance of learning a policy that satisfies the LTL formula $\mathsf{F}\,\text{goal}$.

Further, the problem is not limited to this formula. For example, suppose the state h has water, then the objective "never step on water", expressed as the formula $\mathsf{G}\,\neg\text{water}$, has the same problem in these two MDPs. Therefore, for the full objective **o1**, if the gridworld is one of these two MDPs in Figure 4.1, the agent cannot guarantee to learn a behavior that satisfies the objective. More generally, for any LTL formula describing an infinite-horizon property, I construct two counterexample MDPs with the same nature as the ones in Figure 4.1, and prove that it is impossible to guarantee learning the optimal policy.

4.1.3 Finitary LTL Objective Example

The complement of infinite-horizon objectives are *finitary objectives* — objectives that require inspecting only a finite number of trajectory steps to determine if the trajectory satisfies the objective. For example, the objective **o2** "eventually reach the goal and step on the water in at most 15 steps" is finitary since it requires inspecting only the first 15 steps of a trajectory to determine if the trajectory satisfies the objective. In this chapter, I show that these objectives are learnable — there are algorithms that guarantee learning a near-optimal behavior for these objectives with high probability in a finite amount of interactions with the environment and computation.

Consider the example objective **o2**, and I send an agent into the gridworld of unknown dynamics. As mentioned, there are algorithm that guarantee learning a near-optimal behavior for this objective, which I outline below.

First, I construct a finite-horizon cumulative reward objective equivalent to the given finitary objective. Specifically, I define a reward function that assigns a 1 if the agent reaches the goal within at most 15 steps while avoiding stepping on water during this time; otherwise, it assigns a reward of 0.

Since this reward function depends on the sequence of the first 15 steps, it operates on an augmented state space that tracks the last 15 steps of the agent's trajectory. This transformation results in a new MDP, where each state represents a sequence of 15 states from the original gridworld MDP.

As I will show in the main chapter, a near-optimal policy for this constructed reward objective in the new MDP is also near-optimal for the original finitary objective in the original gridworld MDP. This equivalence allows the use of standard reinforcement learning algorithms designed for finite-horizon cumulative reward settings. For example, a practitioner can apply the ORLC algorithm from Dann et al. [38], which provides learning guarantees for finite-horizon objectives. By learning an optimal policy in this new MDP, the agent also learns a near-optimal strategy for the original finitary objective.

4.1.4 Prior Works

Despite the developments on reinforcement learning with LTL objectives, the infinite-horizon nature of LTL objectives presents challenges that have been alluded to — but not formally treated — in prior work. Henriques et al. [18], Ashok, Křetínský, and Weininger [19], and Jiang et al. [20] noted slow learning times for mastering infinite-horizon properties. Littman et al. [21] provided a specific environment that illustrates the intractability of learning for a specific infinite-horizon objective, arguing for using a discounted LTL variant.

A similar issue exists for infinite-horizon, average-reward objectives. Specifically, reinforcement-learning algorithms are known to lack guarantees on the learned behavior for infinite-horizon, average-reward objectives without additional assumptions on the MDP [33].

However, no prior work has formally analyzed the learnability of LTL objectives.[1]

I leverage the PAC-MDP framework [11] to prove that reinforcement learning for infinite-horizon LTL objectives is intractable. The intuition for this intractability is: Any finite number of interactions with an environment with unknown transition dynamics is insufficient

[1]Concurrent to this work, Alur et al. [27] also examines the intractability of LTL objectives. They state and prove a theorem that is a weaker version of the core theorem of this work. Their work was made public while this work was under conference review. I discuss their work in Section 4.6.

to identify the environment dynamics perfectly. Moreover, for an infinite-horizon objective, a behavior's satisfaction probability under the inaccurate environment dynamics can be arbitrarily different from the behavior's satisfaction probability under the true dynamics. Consequently, a learner cannot confidently guarantee that it has identified near-optimal behavior for an infinite-horizon objective.

4.1.5 Implications for Relevant and Future Work

My results provide a framework for categorizing approaches that either focus on tractable LTL objectives or weaken an algorithm's guarantees. As a result, I interpret several previous approaches as instantiations of the following categories:

- Work with finite-horizon LTL objectives, the complement of infinite-horizon objectives, to obtain guarantees on the learned behavior [18]. These objectives, like $a \wedge \mathsf{X} a$ (a is true for two steps), are decidable within a known finite number of steps.

- Seek a best-effort confidence interval [19]. Specifically, the interval can be trivially large in the worst case, denoting that learned behavior is a maximally poor approximation of the optimal behavior.

- Make additional assumptions about the environment to obtain guarantees on the learned behavior [14, 35].

- Change the problem by working with LTL-like objectives such as: 1. relaxed LTL objectives that become exactly LTL in the (unreachable) limit [1, 2, 16, 17] and 2. objectives that use temporal operators but employ a different semantics [15, 21, 23, 24]. The learnability of these objectives is a potential future research direction.

4.2 Linear Temporal Logic Objectives

This section describes LTL and its use in objectives.

4.2.1 Linear Temporal Logic

A linear temporal logic (LTL) formula is built from a finite set of atomic propositions Π, logical connectives $\neg, \wedge, \vee$, temporal next X, and temporal operators G (always), F (eventually), and U (until). Equation (4.1) gives the grammar of an LTL formula ϕ over the set

of atomic propositions Π:

$$\phi := a \;\Big|\; \neg\phi \;\Big|\; \phi \wedge \phi \;\Big|\; \phi \vee \phi \;\Big|\; \mathsf{X}\phi \;\Big|\; \mathsf{G}\,\phi \;\Big|\; \mathsf{F}\,\phi \;\Big|\; \phi\,\mathsf{U}\,\phi,\ a \in \Pi. \tag{4.1}$$

LTL is a logic over infinite-length words. Informally, these temporal operators have the following meanings:

- $\mathsf{X}\phi$: the sub-formula ϕ is true in the next time step.
- $\mathsf{G}\,\phi$: the sub-formula ϕ is always true in all future time steps.
- $\mathsf{F}\,\phi$: the sub-formula ϕ is eventually true in some future time steps.
- $\phi\,\mathsf{U}\,\psi$: the sub-formula ϕ is always true until the sub-formula ψ eventually becomes true, after which ϕ is allowed to become false.

Formally, I write $w \vDash \phi$ to denote that the infinite word w satisfies ϕ. Definition 4.2.1 defines this relation.

Definition 4.2.1 (LTL Semantics)**.**

$$\begin{aligned}
&w \vDash a && \textit{iff } a \in w[i] \quad a \in \Pi \\
&w \vDash \neg\phi && \textit{iff } w \nvDash \phi \\
&w \vDash \phi \wedge \psi && \textit{iff } w \vDash \phi \textit{ and } w \vDash \psi \\
&w \vDash \phi \vee \psi && \textit{iff } w \vDash \phi \textit{ or } w \vDash \psi \\
&w \vDash \mathsf{X}\phi && \textit{iff } w[1{:}] \vDash \phi \\
&w \vDash \mathsf{G}\,\phi && \textit{iff } \forall i \geq 0, w[i{:}] \vDash \phi \\
&w \vDash \mathsf{F}\,\phi && \textit{iff } \exists i \geq 0, w[i{:}] \vDash \phi \\
&w \vDash \phi\,\mathsf{U}\,\psi && \textit{iff } \exists j \geq 0, w[j{:}] \vDash \psi \textit{ and } \forall k.\, 0 \leq k < j \Rightarrow w[k{:}] \vDash \phi.
\end{aligned}$$

The rest of the operators are defined as syntactic sugar in terms of operators in Definition 4.2.1 as: $\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$, $\mathsf{F}\,\phi \equiv \text{True}\,\mathsf{U}\,\phi$, $\mathsf{G}\,\phi \equiv \neg\mathsf{F}\,\neg\phi$. For a more in-depth introduction of LTL, I refer readers to Baier and Katoen [42].

4.2.2 MDP with LTL Objectives

An LTL objective maximizes the probability of satisfying an LTL formula. I formalize this notion below.

An *LTL specification* for an MDP is a tuple $(\mathcal{L}, \phi)$, where $\mathcal{L}\colon S \to 2^{\Pi}$ is a labeling function, and ϕ is an LTL formula over atomic propositions Π. The labeling function is a

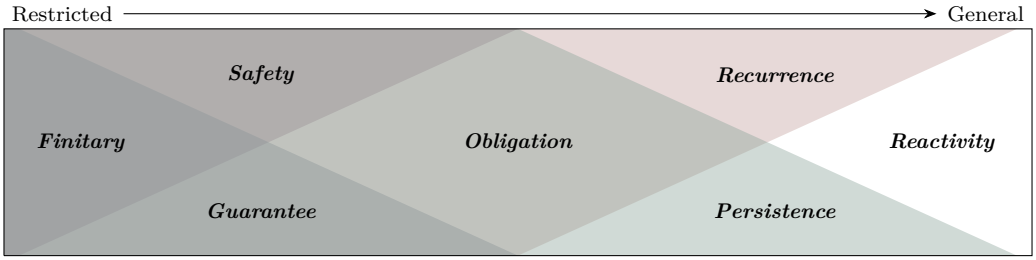


Figure 4.2: The hierarchy of LTL

classifier mapping each MDP state to a tuple of truth values of the atomic propositions in ϕ. For a sample path w, I use $\mathcal{L}(w)$ to denote the element-wise application of $\mathcal{L}$ on w.

The LTL objective ξ specified by the LTL specification is the satisfaction of the formula ϕ of a sample path mapped by the labeling function $\mathcal{L}$, that is: $\kappa(w) \triangleq \mathbb{1}\{\mathcal{L}(w) \vDash \phi\}$. The value of this objective is called the *satisfaction probability* of ξ:

$$V^{\pi}_{\mathcal{M},\xi} = \mathsf{P}_{w\sim\mathcal{D}}(\mathcal{L}(w) \vDash \phi) \quad (\mathcal{D} \text{ induced by } \mathcal{M} \text{ and } \pi).$$

4.2.3 Infinite Horizons in LTL Objectives

An LTL formula describes either a finite-horizon or infinite-horizon property over a infinite-length word. Manna and Pnueli [43] classified LTL formulas into seven classes, as shown in Figure 4.2. Each class includes all the classes to the left of that class: For example, *Finitary* $\subset$ *Guarantee*, but *Safety* $\not\subset$ *Guarantee*. The *Finitary* class is the most restricted and the *Reactivity* class is the most general.

I describe the key properties of all classes in the LTL hierarchy. Among these classes, the classes *Finitary*, *Guarantee*, and *Safety* are particularly relevant to this work.

- $\phi \in$ *Finitary* iff there exists a horizon H such that infinite-length words sharing the same prefix of length H are either all accepted or all rejected by ϕ. For example, the formula $a \wedge \mathsf{X}a$ (that is, a is true for two steps) is in *Finitary*.

- $\phi \in$ *Guarantee* iff there exists a language of finite words L (that is, a Boolean function on finite-length words) such that $w \vDash \phi$ if L accepts a prefix of w. Informally, a formula in *Guarantee* asserts that something eventually happens. For example, the formula $\mathsf{F}\,a$ (that is, eventually a is true) is in *Guarantee*.

- $\phi \in$ *Safety* iff there exists a language of finite words L such that $w \vDash \phi$ if L accepts all prefixes of w. Informally, a formula in *Safety* asserts that something always happens. For example, the formula $\mathsf{G}\,a$ (that is, a is always true) is in *Safety*.

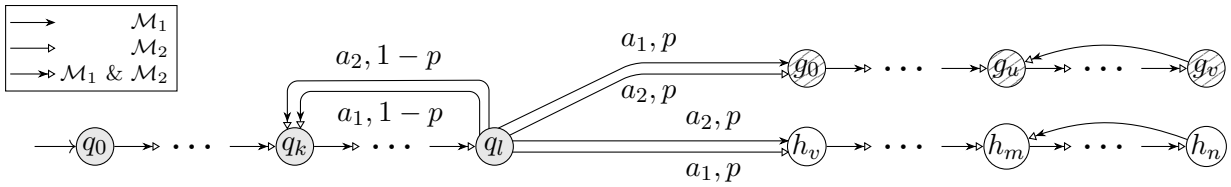


Figure 4.3: Counterexample MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$, with transitions distinguished by arrow types (see legend). Both MDPs are parameterized by the parameter p that is in range $0 < p < 1$. Unlabeled edges are deterministic (actions a_1 and a_2 transition with probability 1). Ellipsis indicates a deterministic chain of states.

- $\phi \in$ *Obligation* iff ϕ is a logical combination of formulas in *Guarantee* and *Safety*. For example, the formula $\mathsf{F}\, a \wedge \mathsf{G}\, b$ is in *Obligation*.
- $\phi \in$ *Persistence* iff there exists a language of finite words L such that $w \vDash \phi$ if L accepts all but finitely many prefixes of w. Informally, a formula in *Persistence* asserts that something happens finitely often. For example, the formula $\mathsf{F}\,\mathsf{G}\, a$ (that is, a is not true for only finitely many times, and eventually a stays true forever) is in *Persistence*.
- $\phi \in$ *Recurrence* iff there exists a language of finite words L such that $w \vDash \phi$ if L accepts infinitely many prefixes of w. Informally, a formula in *Recurrence* asserts that something happens infinitely often. For example, the formula $\mathsf{G}\,\mathsf{F}\, a$ (that is, a is true for infinitely many times) is in *Recurrence*.
- $\phi \in$ *Reactivity* iff ϕ is a logical combination of formulas in *Recurrence* and *Persistence*. For example, the formula $\mathsf{G}\,\mathsf{F}\, a \wedge \mathsf{F}\,\mathsf{G}\, b$ is in *Reactivity*.

Moreover, the set of finitary is the intersection of the set of guarantee formulas and the set of safety formulas. Any $\phi \in$ *Finitary*, or equivalently $\phi \in$ *Guarantee* $\cap$ *Safety*, inherently describes finite-horizon properties. Any $\phi \notin$ *Finitary*, or equivalently $\phi \in$ *Guarantee*$^{\complement} \cup$ *Safety*$^{\complement}$, inherently describes infinite-horizon properties. I will show that reinforcement-learning algorithms cannot provide PAC guarantees for LTL objectives specified by formulas that describe infinite-horizon properties.

4.3 Learnability of LTL Objectives

This section states and outlines the proof to the main result.

By specializing the κ-PAC definitions (Definitions 3.6.1 and 3.6.2) with the definition of LTL objectives in Section 4.2.2, I obtain the following definitions of LTL-PAC.

Definition 4.3.1. *Given an LTL objective ξ, a planning-with-generative-model algorithm $(\mathcal{A}^S, \mathcal{A}^L)$ is* LTL-PAC *(probably approximated correct for LTL objective ξ) in an environment MDP $\mathcal{M}$ for the LTL objective ξ if, with the sequence of transitions T of length N sampled using the sampling algorithm $\mathcal{A}^S$, the learning algorithm $\mathcal{A}^L$ outputs a non-Markovian ϵ-optimal policy with a probability of at least $1-\delta$ for all $\epsilon > 0$ and $0 < \delta < 1$. That is,*

$$\mathsf{P}_{\mathsf{T}\sim\langle\mathcal{M},\mathcal{A}^S\rangle_N}\left(V^{\mathcal{A}^L(T)}_{\mathcal{M},\xi} \geq V^{\pi^*}_{\mathcal{M},\xi} - \epsilon\right) \geq 1-\delta. \tag{4.2}$$

The probability on the left of the inequality is the *LTL-PAC probability* of the algorithm $(\mathcal{A}^{\mathrm{S}}, \mathcal{A}^{\mathrm{L}})$.

Definition 4.3.2. *Given an LTL objective ξ, an LTL-PAC planning-with-generative-model algorithm for ξ is* sample efficiently *LTL-PAC if the number of sampled transitions N is asymptotically polynomial to $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $|S|$, $|A|$.*

4.3.1 The Main Theorem

With the above definitions, I can now define the PAC learnability of an LTL objective and state the main theorem.

Definition 4.3.3. *An LTL formula ϕ over atomic propositions Π is* LTL-PAC-learnable by planning-with-generative-model (reinforcement-learning) *if there exists a sample efficiently LTL-PAC planning-with-generative-model (reinforcement-learning) algorithm for all environment MDPs and all consistent labeling functions $\mathcal{L}$ (that is, $\mathcal{L}$ maps from the MDP's states to 2^{Π}) for the LTL objective specified by $(\mathcal{L}, \phi)$.*

Theorem 4.3.4. *An LTL formula ϕ is LTL-PAC-learnable by reinforcement-learning (planning-with-generative-model) if (and only if) ϕ is finitary.*

Between the two directions of Theorem 4.3.4, the forward direction, "only if", is more important. The forward direction states that for any LTL formula not in *Finitary* (that is, infinite-horizon properties), there does not exist a planning-with-generative-model algorithm — which by definition also excludes any reinforcement-learning algorithm — that is sample efficiently LTL-PAC for all environments. This result is the core contribution of the paper — infinite-horizon LTL formulas are not sample efficiently LTL-PAC-learnable.

Alternatively, the reverse direction of Theorem 4.3.4 states that, for any finitary formula (finite-horizon properties), there exists a reinforcement-learning algorithm—which by definition is also a planning-with-generative-model algorithm—that is sample efficiently LTL-PAC for all environments.

4.3.2 Consequence of the Theorem

Theorem 4.3.4 implies that: For any non-finitary LTL objective, given any arbitrarily large finite sample of transitions, the learned policy need not perform near-optimally. This implication is unacceptable in applications that require strong guarantees of the system's behavior.

4.3.3 Proof of Theorem 4.3.4: Forward Direction

This section proves the forward direction of Theorem 4.3.4. First, I construct a family of pairs of MDPs. Then, for the singular case of the LTL formula $\mathsf{F}\, g_0$, I derive a sample complexity lower bound for any LTL-PAC planning-with-generative-model algorithm applied to my family of MDPs. This lower bound necessarily depends on a specific transition probability in the MDPs. Finally, I generalize this bound to any non-finitary LTL formula and conclude the proof.

4.3.3.1 MDP Family

I give two constructions of parameterized counterexample MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$ shown in Figure 4.3. The key design behind each pair in the family is that no planning-with-generative-model algorithm can learn a policy that is simultaneously ϵ-optimal on both MDPs without observing a number of samples that depends on the probability of a specific transition.

Both MDPs are parameterized by the shape parameters k, l, u, v, m, n, and an unknown transition probability parameter p. The actions are $\{a_1, a_2\}$, and the state space is partitioned into three regions (as shown in Figure 4.3: states $q_{0\ldots l}$ (the grey states), states $g_{0\ldots v}$ (the line-hatched states), and states $h_{0\ldots n}$ (the white states). All transitions, except $q_l \to g_0$ and $q_l \to h_0$, are the same between $\mathcal{M}_1$ and $\mathcal{M}_2$. The effect of this difference between the two MDPs is that, for $\mathcal{M}_i$, $i \in \{1, 2\}$:

- Action a_i in $\mathcal{M}_i$ at the state q_l will transition to the state g_0 with probability p, inducing a run that cycles in the region $g_{u\ldots v}$ forever.

- Action a_{3-i} (the alternative to a_i) in $\mathcal{M}_i$ at the state q_l will transition to the state h_0 with probability p, inducing a run that cycles in the region $h_{m\ldots n}$ forever.

Further, for any policy, a run of the policy on both MDPs must eventually reach g_0 or h_0 with probability 1, and ends in an infinite cycle in either $g_{u\ldots v}$ or $h_{m\ldots n}$.

4.3.3.2 Sample Complexity of $\mathsf{F}\, g_0$

I next consider the LTL objective ξ^{g_0} specified by the LTL formula $\mathsf{F}\, g_0$ and the labeling function $\mathcal{L}^{g_0}$ that labels only the state g_0 as *true*. A sample path on the MDPs (Figure 4.3) satisfies this objective iff the path reaches the state g_0.

Given $\epsilon > 0$ and $0 < \delta < 1$, my goal is to derive a lower bound on the number of sampled environment transitions performed by an algorithm, so that the satisfaction probability of π, the learned policy, is ϵ-optimal (i.e., $V^{\pi}_{\mathcal{M},\xi^{g_0}} \geq V^{\pi^*}_{\mathcal{M},\xi^{g_0}} - \epsilon$) with a probability of least $1-\delta$.

The key rationale behind the following lemma is that, if a planning-with-generative-model algorithm has not observed any transition to either g_0 or h_0, the learned policy cannot be ϵ-optimal in both $\mathcal{M}_1$ and $\mathcal{M}_2$.

Lemma 4.3.5. *For any planning-with-generative-model algorithm $(\mathcal{A}^S, \mathcal{A}^L)$, it must be the case that:* $\min(\zeta_1, \zeta_2) \leq \frac{1}{2}$, *where* $\zeta_i = \mathsf{P}_{\mathsf{T}}\left(V^{\mathcal{A}^L(T)}_{\mathcal{M}_i,\xi^{g_0}} \geq V^{\pi^*}_{\mathcal{M}_i,\xi^{g_0}} - \epsilon \;\middle|\; n(T) = 0 \right)$ *and $n(T)$ is the number of transitions in T that start from q_l and end in either g_0 or h_0.*

The value ζ_i is the LTL-PAC probability of a learned policy on $\mathcal{M}_i$, given that the planning-with-generative-model algorithm did not observe any information that allows the algorithm to distinguish between $\mathcal{M}_1$ and $\mathcal{M}_2$.

Proof. I present a proof of Lemma 4.3.5 in Section 4.7.1. □

A planning-with-generative-model algorithm cannot learn an ϵ-optimal policy without observing a transition to either g_0 or h_0. Thus, I bound the sample complexity of the algorithm from below by the probability that the sampling algorithm observes such a transition:

Lemma 4.3.6. *For the LTL objective ξ^{g_0}, the number of samples, N, for an LTL-PAC planning-with-generative-model algorithm for both $\mathcal{M}_1$ and $\mathcal{M}_2$ (for any instantiation of the parameters k, l, u, v, m, n) has a lower bound of* $N \geq \frac{\log(2\delta)}{\log(1-p)}$.

Below I give a proof sketch of Lemma 4.3.6; I give the complete proof in Section 4.7.3.

Proof Sketch of Lemma 4.3.6. First, I assert that the two inequalities of Equation (4.2) for both $\mathcal{M}_1$ and $\mathcal{M}_2$ holds true for a planning-with-generative-model algorithm. Next, by conditioning on $n(T) = 0$, plugging in the notation of ζ_i, and relaxing both inequalities, I get $(1-\zeta_i)\mathsf{P}_{\mathsf{T}}(n(T) = 0) \leq \delta$, for $i \in \{1, 2\}$. Then, since $n(T) = 0$ only occurs when all transitions from q_l end in q_k, I have $\mathsf{P}_{\mathsf{T}}(n(T) = 0) \geq (1-p)^N$. Combining the inequalities, I get $(1 - \min(\zeta_1, \zeta_2))(1-p)^N \leq \delta$. Finally, I apply Lemma 4.3.5 to get the desired lower bound of $N \geq \frac{\log(2\delta)}{\log(1-p)}$. □

4.3.3.3 Sample Complexity of Non-finitary Formulas

This section generalizes my lower bound on $\mathsf{F}\, g_0$ to all non-finitary LTL formulas. The key observation is that for any non-finitary LTL formula, I can choose a pair of MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, from my MDP family. For both MDPs in this pair, finding an ϵ-optimal policy for $\mathsf{F}\, g_0$ is reducible to finding an ϵ-optimal policy for the given formula. By this reduction, the established lower bound for the case of $\mathsf{F}\, g_0$ also applies to the case of any non-finitary formula. Therefore, the sample complexity of learning an ϵ-optimal policy for any non-finitary formula has a lower bound of $\frac{\log(2\delta)}{\log(1-p)}$.

I will use $[w_1; w_2; \dots w_n]$ to denote the concatenation of the finite-length words $w_1 \dots w_n$. I will use w^i to denote the repetition of the finite-length word w by i times, and w^ω to denote the infinite repetition of w.

Definition 4.3.7. *An accepting (resp. rejecting) infinite-length word $[w_a; w_b^\omega]$ of ϕ is* uncommittable *if there exists finite-length words w_c, w_d such that ϕ rejects (resp. accepts) $[w_a; w_b^i; w_c; w_d^\omega]$ for all $i \in \mathbb{N}$.*

Lemma 4.3.8. *If ϕ has an uncommittable word w, there is an instantiation of $\mathcal{M}_1$ (or $\mathcal{M}_2$) in Figure 4.3 and a labeling function $\mathcal{L}$, such that, for any policy, the satisfaction probabilities of that policy in $\mathcal{M}_1$ (or $\mathcal{M}_2$) for the LTL objectives specified by $(\mathcal{L}, \phi)$ and $(\mathcal{L}^{g_0}, \mathsf{F}\, g_0)$ are the same.*

Proof. For an uncommittable word w, I first find the finite-length words w_a,w_b,w_c,w_d according to Definition 4.3.7. I then instantiate $\mathcal{M}_1$ and $\mathcal{M}_2$ in Figure 4.3 as follows.

- If w is an uncommittable accepting word, I set k, l, u, v, m, n (Figure 4.3) to $|w_a|$, $|w_a| + |w_b|$, 0, $|w_b|$, $|w_c|$ and $|w_c| + |w_d|$, respectively. I then set the labeling function as in Equation (4.4).
- If w is an uncommittable rejecting word, I set k, l, u, v, m, n (Figure 4.3) to $|w_a|$, $|w_a| + |w_b|$, $|w_c|$, $|w_c| + |w_d|$, 0 and $|w_b|$, respectively. I then set the labeling function as in Equation (4.5).

$$\mathcal{L}(s) = \begin{cases} [w_a; w_b][j] & \text{if } s{=}q_j \\ w_b[j] & \text{if } s{=}g_j \\ [w_c; w_d][j] & \text{if } s{=}h_j \end{cases} \qquad (4.4)$$

$$\mathcal{L}(s) = \begin{cases} [w_a; w_b][j] & \text{if } s{=}q_j \\ [w_c; w_d][j] & \text{if } s{=}g_j \\ w_b[j] & \text{if } s{=}h_j \end{cases} \qquad (4.5)$$

In words, for an uncommittable accepting word, I label the states $q_{0\dots l}$ one-by-one by $[w_a; w_b]$; I label the states $g_{0\dots v}$ one-by-one by w_b (and set $u = 0$, which eliminates the chain of states $g_{0\dots u}$); I label the states $h_{0\dots n}$ one-by-one by $[w_c; w_d]$. Symmetrically, for an uncommittable rejecting word, I label the states $q_{0\dots l}$ one-by-one by $[w_a; w_b]$; I label the states

$g_{0\ldots v}$ one-by-one by $[w_c; w_d]$; I label the states $h_{0\ldots n}$ one-by-one by w_b (and set $m = 0$, which eliminates the chain of states $h_{0\ldots m}$).

By the above instantiation, the two objectives specified by $(\mathcal{L}, \phi)$ and $(\mathcal{L}^{g_0}, \mathsf{F}\, g_0)$ are equivalent in $\mathcal{M}_1$ and $\mathcal{M}_2$. In particular, any path in $\mathcal{M}_1$ or $\mathcal{M}_2$ satisfies the LTL objective specified by $(\mathcal{L}, \phi)$ if and only if the path visits the state g_0 and therefore also satisfies the LTL objective specified by $(\mathcal{L}^{g_0}, \mathsf{F}\, g_0)$. Therefore, any policy must have the same satisfaction probability for both objectives. □

Lemma 4.3.9. *For $\phi \notin$ Finitary, the number of samples for a planning-with-generative-model algorithm to be LTL-PAC has a lower bound of $N \geq \frac{\log(2\delta)}{\log(1-p)}$.*

Proof. A corollary of Lemma 4.3.8 is: for any ϕ that has an uncommittable word, I can construct a pair of MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$ in the family of pairs of MDPs in Figure 4.3, such that, in both MDPs, a policy is sample efficiently LTL-PAC for the LTL objective specified by $(\mathcal{L}, \phi)$ if it is sample efficiently LTL-PAC for the LTL objective specified by $(\mathcal{L}^{g_0}, \mathsf{F}\, g_0)$. This property implies that the lower bound in Lemma 4.3.6 for the objective specified by $(\mathcal{L}^{g_0}, \mathsf{F}\, g_0)$ also applies to the objective specified by $(\mathcal{L}, \phi)$, provided that any $\phi \notin$ *Finitary* has an uncommittable word. In Section 4.7.4, I prove a lemma that any formula $\phi \notin$ *Guarantee* has an uncommittable accepting word, and any formula $\phi \notin$ *Safety* has an uncommittable rejecting word. Since *Finitary* is the intersection of *Guarantee* and *Safety*, this completes the proof. □

4.3.3.4 Conclusion

Note that the lower bound $\frac{\log(2\delta)}{\log(1-p)}$ depends on p, the transition probability in the constructed MDPs. Moreover, for $\delta < \frac{1}{2}$, as p approaches 0, this lower bound goes to infinity. As a result, the bound does not satisfy the definition of sample efficiently LTL-PAC planning-with-generative-model algorithm for the LTL objective (Definition 3.6.2), and thus no algorithm is sample efficiently LTL-PAC. Therefore, LTL formulas not in *Finitary* are not LTL-PAC-learnable. This completes the proof of the forward direction of Theorem 4.3.4.

4.3.4 Proof Sketch of Theorem 4.3.4: Reverse Direction

This section gives a proof sketch to the reverse direction of Theorem 4.3.4. I give a complete proof in Section 4.3.5.

I prove the reverse direction of Theorem 4.3.4 by reducing the problem of learning a policy for any finitary formula to the problem of learning a policy for a finite-horizon cumulative rewards objective. I conclude the reverse direction of the theorem by invoking a known PAC reinforcement-learning algorithm on the later problem.

- **Reduction to Infinite-horizon Cumulative Rewards.** First, given an LTL formula in *Finitary* and an environment MDP, I will construct an *augmented MDP with rewards* similar to Camacho et al. [23] and Giacomo et al. [24]. I reduce the problem of finding the optimal non-Markovian policy for satisfying the formula in the original MDP to the problem of finding the optimal Markovian policy that maximizes the infinite-horizon (undiscounted) cumulative rewards in this augmented MDP.

- **Reduction to Finite-horizon Cumulative Rewards.** Next, I reduce the infinite-horizon cumulative rewards to a finite-horizon cumulative rewards, using the fact that the formula is finitary.

- **Sample Complexity Upper Bound.** Lastly, Dann et al. [38] have derived an upper bound on the sample complexity for a reinforcement-learning algorithm for finite-horizon MDPs. I thus specialize this known upper bound to my problem setup of the augmented MDP and conclude that any finitary formula is PAC-learnable.

4.3.5 Proof of Theorem 4.3.4: the Reverse Direction

In this section, I give a proof to the reverse direction of Theorem 4.3.4.

Reduction to Infinite-horizon Cumulative Rewards Given an LTL formula ϕ in *Finitary* with atomic propositions Π, one can compile ϕ into a DFA $\bar{\mathcal{M}} = (\bar{S}, 2^{\Pi}, \bar{P}, \bar{s_0}, \bar{s_{\text{acc}}})$ that decides the satisfaction of ϕ [44]. In particular, for a given sample path w of Markov chain induced by a policy and the environment MDP, $\mathcal{L}(w)$ satisfies ϕ if and only if the DFA, upon consuming $\mathcal{L}(w)$, eventually reaches the accept state s_{acc}. Here, the DFA has a size (in the worst case) doubly exponential to the size of the formula: $|\bar{S}| = \mathcal{O}(2\text{exp}(|\phi|))$ [45].

I then use the following product construction to form an augmented MDP with rewards $\hat{\mathcal{M}} = (\hat{S}, \hat{A}, \hat{P}, \hat{s_0}, \hat{R})$. Specifically,

- The states and actions are: $\hat{S} = S \times \bar{S}$ and $\hat{A} = A$.

- The transitions follow the transitions in the environment MDP and the DFA simultaneously, where the action input of the DFA come from labeling the current state of the environment MDP. In particular, the transitions in the augmented MDP follows the equations: $\hat{P}((s, \bar{s}), a, (s', \bar{s}')) = P(s, a, s')$ and $\bar{s}' = \bar{P}(\bar{s}, \mathcal{L}(s))$.

- The reward function assigns a reward of one to any transition from a non-accepting state that reaches s_{acc} in the DFA, and zero otherwise:

$$R((s, \hat{s}), a, (s', \hat{s}')) = \mathbb{1}\{s \neq s_{\text{acc}} \wedge \hat{s}' = s_{\text{acc}}\}.$$

By construction, each run of the augmented MDP gives a reward of 1 iff the run satisfies the finitary formula ϕ. The expected (undiscounted) infinite-horizon cumulative rewards thus equals the satisfaction probability of the formula. Therefore, maximizing the infinite-horizon cumulative rewards in the augmented MDP is equivalent to maximizing the satisfaction probability of ϕ in the environment MDP.

Reduction to Finite-horizon Cumulative Rewards By the property of LTL hierarchy [43], for any LTL formula ϕ in *Finitary* and an infinite-length word w, one can decide if ϕ accepts w by inspecting a length-H prefix of w. Here, H is a constant that is computable from ϕ. In particular, H equals the longest distance from the start state to a terminal state in my constructed DFA. [2] Thus, since the product construction above does not assign any reward after the horizon H, the infinite-horizon cumulative rewards is further equivalent to the finite-horizon (of length H) cumulative rewards. Therefore, finding the optimal policy for ϕ is equivalent to finding the optimal policy that maximizes the cumulative rewards for a finite horizon H in the augmented MDP.

Sample Complexity Upper Bound Lastly, Dann et al. [38] gave a reinforcement-learning algorithm for finite-horizon cumulative rewards called ORLC (optimistic reinforcement learning with certificates). The ORLC algorithm is sample efficiently PAC [3] and has a sample complexity of $\tilde{\mathcal{O}}\left(\left(\frac{|S||A|H^3}{\epsilon^2} + \frac{|S|^2|A|H^4}{\epsilon}\right)\log\frac{1}{\delta}\right)$. [4] Incorporating the fact that the augmented MDP has $|\hat{S}| = |S| \cdot \mathcal{O}(2\exp(|\phi|))$ number of states, I obtain a sample complexity upper bound of $\tilde{\mathcal{O}}\left(\left(\frac{|S|\,2\exp|\phi||A|H^3}{\epsilon^2} + \frac{|S|^2(2\exp|\phi|)^2|A|H^4}{\epsilon}\right)\log\frac{1}{\delta}\right)$ for the overall reinforcement-learning algorithm.

Since for any finitary formula, I have constructed a reinforcement-learning algorithm that is sample efficiently LTL-PAC for all environment MDPs, this concludes my proof that any finitary formula is LTL-PAC-learnable.

4.4 Empirical Justifications

This section empirically demonstrates my main result, the forward direction of Theorem 4.3.4.

[2] Note that since ϕ is finitary, the DFA does not have any cycles except at the terminal states [46].

[3] The ORLC algorithm provides a guarantee called individual policy certificates (IPOC) bound. Dann et al. [38] showed that this guarantee implies my PAC definition, which they called a supervised-learning style PAC bound. Therefore, the ORLC algorithm is a PAC reinforcement-learning algorithm for finite-horizon cumulative rewards by my definition.

[4] The notation $\tilde{\mathcal{O}}(.)$ is the same as $\mathcal{O}(.)$, but ignores any log-terms. The bound given by Dann et al. [38] is $\tilde{\mathcal{O}}\left(\frac{|S|^2|A||H^2}{\epsilon^2}\log\frac{1}{\delta}\right)$. It is a upper bound on the number of episodes. To make the bound consistent with my lower bound on the number of sampled transitions, I multiply it by an additional H term.

Previous work has introduced various reinforcement-learning algorithms for LTL objectives [1, 2, 16, 17]. I ask the research question: *Do the sample complexities of these algorithms depend on the transition probabilities of the environment?* To answer the question, we evaluate various algorithms and empirically measure the sample sizes for them to obtain near-optimal policies with high probability.

4.4.1 Methodology

I consider various recent reinforcement-learning algorithms for LTL objectives [1, 2, 16]. I consider two pairs of LTL formulas and environment MDPs (LTL-MDP pair). The first pair is the formula $\mathsf{F}\, h$ and the counterexample MDP as shown in Figure 4.1. The second pair is adapted from a case study in Sadigh et al. [2]. I focus on the first pair in this section and defer the complete evaluation to Appendix B.

I run the considered algorithms on each chosen LTL-MDP pair with a range of values for the parameter p and let the algorithms perform N environment samples. For each algorithm and each pair of values of p and N, I fix $\epsilon = 0.1$ and repeatedly run the algorithm to obtain a Monte Carlo estimation of the LTL-PAC probability (left side of Equation (4.2)) for that setting of p, N and ϵ. I repeat each setting until the estimated standard deviation of the estimated probability is within 0.01. In the end, for each algorithm and LTL-MDP pair I obtain $5 \times 21 = 105$ LTL-PAC probabilities and their estimated standard deviations.

For the first LTL-MDP pair, I vary p by a geometric progression from 10^{-1} to 10^{-3} in 5 steps. I vary N by a geometric progression from 10^1 to 10^5 in 21 steps. For the second LTL-MDP pair, I vary p by a geometric progression from 0.9 to 0.6 in 5 steps. I vary N by a geometric progression from 3540 to 9×10^4 in 21 steps. If an algorithm does not converge to the desired LTL-PAC probability within 9×10^4 steps, I rerun the experiment with an extended range of N from 3540 to 1.5×10^5.

4.4.2 Results

Figure 4.4 presents the results for the algorithm in Bozkurt et al. [1] with the setting of Multi-discount, Q-learning, and the first LTL-MDP pair. On the left, I plot the LTL-PAC probabilities vs. the number of samples N, one curve for each p. On the right, I plot the intersections of the curves in the left plot with a horizontal cutoff of 0.9.

As I see from the left plot of Figure 4.4, for each p, the curve starts at 0 and grows to 1 in a sigmoidal shape as the number of samples increases. However, as p decreases, the MDP becomes harder: As shown on the right plot of Figure 4.4, the number of samples required to reach the particular LTL-PAC probability of 0.9 grows exponentially. Results for other

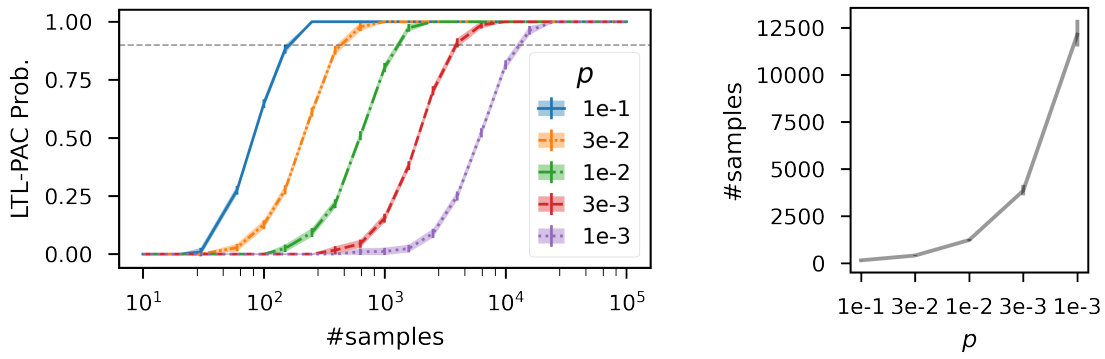


Figure 4.4: Left: LTL-PAC probabilities vs. number of samples, varying parameters p. Right: number of samples needed to reach 0.9 LTL-PAC probability vs. parameter p.

algorithms, environments and LTL formulas are similar and lead to the same conclusion.

Figure B.3 presents the complete results for all settings for the first LTL-MDP pair, and Figure B.4 present the complete results for all settings for the second LTL-MDP pair. These results are similar and lead to the same analysis as above.

4.4.3 Results Interpretation

Since the transition probabilities (p in this case) are unknown in practice, one can't know which curve in the left plot a given environment will follow. Therefore, given any finite number of samples, these reinforcement-learning algorithms cannot provide guarantees on the LTL-PAC probability of the learned policy. This result supports Theorem 4.3.4.

4.5 Directions Forward

I have established the intractability of reinforcement learning for infinite-horizon LTL objectives. Specifically, for any infinite-horizon LTL objective, the learned policy need not perform near-optimally given any finite number of environment interactions. This intractability is undesirable in applications that require strong guarantees, such as traffic control, robotics, and autonomous vehicles [47–49].

Going forward, I categorize approaches that either focus on tractable objectives or weaken the guarantees required by an LTL-PAC algorithm. I obtain the first category from the reverse direction of Theorem 4.3.4, and each of the other categories by relaxing a specific requirement that Theorem 4.3.4 places on an algorithm. Further, I classify previous approaches into these categories.

4.5.1 Use a Finitary Objective

For these succinct specification languages, by the reduction of these languages to finitary properties and the reverse direction of Theorem 4.3.4, there exist reinforcement-learning algorithms that give LTL-PAC guarantees.

Researchers have introduced specification languages that express finitary properties and have applied reinforcement learning to objectives expressed in these languages.

Henriques et al. [18] introduced a variant of LTL called *Bounded LTL* and used Bounded LTL objective for reinforcement learning. Every Bounded LTL formula is decidable by a bounded length prefix of the input word. Moreover, each Bounded LTL formula is equivalent to an finitary LTL formula. Therefore, I classified this approach as using a Finitary objective.

Jothimurugan, Alur, and Bastani [25] introduced a task specification language over finite-length words. Further, their definition of an MDP contains an additional finite time horizon H. Each sample path of the MDP is then a length-H finite-length word and is evaluated by a formula of the task specification language.[5] Each formula of the task specification language with a fixed finite horizon H is equivalent to an LTL formula in the Finitary class. Therefore, I classified this approach as using a Finitary objective.

One value proposition of these approaches is that they provide succinct specifications because finitary properties written in LTL directly are verbose. For example, the finitary property "a holds for 100 steps" is equivalent to an LTL formula with a conjunction of 100 terms: $a \wedge \mathsf{X}a \wedge \cdots \wedge (\underbrace{\mathsf{X}\ldots\mathsf{X}}_{99 \text{ times}} a)$.

4.5.2 Best-effort Guarantee

The definition of LTL-PAC (Definition 4.3.1) requires a reinforcement-learning algorithm to learn a policy with satisfaction probability within ϵ of optimal, for all $\epsilon > 0$. However, it is possible to relax this quantification over ϵ so that an algorithm only returns a policy with the best-available ϵ it finds.

For example, Ashok, Křetínský, and Weininger [19] introduced a reinforcement-learning algorithm for LTL objectives in the *Guarantee* class. Using a specified time budget, the algorithm returns a policy and an potential error interval ϵ. Notably, it is possible for the returned ϵ to be 1, a vacuous bound on performance.

[5] There are two possible interpretations of the finite horizon in Jothimurugan, Alur, and Bastani [25]. The first interpretation is that the environment MDP inherently terminates and produces length-H sample paths. The second interpretation is that the finite horizon H is part of the specification given by a user of their approach. I used the second interpretation to classify their approach. The difference between the two interpretations is only conceptual — if the environment inherently terminates with a fixed finite horizon H, it would be equivalent to imposing a finite horizon H in the task specification.

4.5.3 Know More About the Environment

The definition of LTL-PAC (Definition 4.3.1) requires a reinforcement-learning algorithm to provide a guarantee for all environments. However, on occasion, one can have prior information on the transition probabilities of the MDP at hand.

For example, Fu and Topcu [14] introduced a reinforcement-learning algorithm with a PAC-MDP guarantee that depends on the time horizon until the MDP reaches a steady state. Given an MDP, this time horizon is generally unknown; however, if one has knowledge of this time horizon *a priori*, it constrains the set of MDPs and yields an LTL-PAC guarantee dependent on this time horizon.

As another example, Brázdil et al. [35] introduced a reinforcement-learning algorithm that provides an LTL-PAC guarantee provided a declaration of the minimum transition probability of the MDP. This constraint, again, bounds the space of considered MDPs.

4.5.4 Use an LTL-like Objective

Theorem 4.3.4 only considers LTL objectives. However, one opportunity for obtaining a PAC guarantee is to change the problem: use an LTL-like specification language that defines similar temporal operators but assigns them a different, less demanding semantics.

4.5.4.1 LTL Surrogate Objectives

One line of work [1, 2, 16, 17] uses LTL formulas as the objective, but also introduces one or more hyper-parameters to relax the formula's semantics. The reinforcement-learning algorithms in these works learn a policy for the environment MDP given fixed values of the hyper-parameters. Moreover, as hyper-parameter values approach a limit point, the learned policy becomes optimal for the hyper-parameter-free LTL formula.[6] The relationship between these relaxed semantics and the original LTL semantics is analogous to the relationship between discounted and average-reward infinite-horizon MDPs. Specifically, the relaxed semantics (resp. discounted MDPs) approaches the LTL semantics (resp. the average-reward MDPs) in the limit of the hyper-parameter values (resp. discount factors) as shown by Bozkurt et al. [1] (resp. Puterman [32]).

I classified Bozkurt et al. [1], Sadigh et al. [2], Hahn et al. [16], and Hasanbeig et al. [17] as using LTL-like objectives, and explained my rationale of these classifications in Section 4.5.

[6] Hahn et al. [16] and Bozkurt et al. [1] showed that there exists a critical setting of the parameters λ^* that produces the optimal policy. However, λ^* depends on the transition probabilities of the MDP and is therefore consistent with my findings.

4.5.4.2 General LTL-like Objectives

Prior approaches [15, 21, 23, 24] also use general LTL-like specifications that do not or are not known to converge to LTL in a limit. For example, Camacho et al. [23] introduced the reward-machine objective that uses a finite state automaton to specify a reward function. As another example, Littman et al. [21] introduced *geometric LTL*. Geometric LTL attaches a geometrically distributed horizon to each temporal operator. The learnability of these general LTL-like objectives is a potential future research direction.

Littman et al. [21] introduced a discounted variant of LTL called *Geometric LTL* (GLTL). A temporal operator in a GLTL formula expires within a time window whose length follows a geometric distribution. For example, a GLTL formula $\mathsf{F}_{0.1}\mathit{goal}$ is satisfied if the sample path reaches the *goal* within a time horizon H, where H follows Geometric(0.1), the geometric distribution with the success parameter 0.1. Since GLTL's semantics is different from LTL's semantics, I classified this approach as using an LTL-like objective.

Li, Vasile, and Belta [15] introduced a LTL variant called *Truncated-LTL* (TLTL). A TLTL formula, like a Bounded LTL formula [18], is decidable by a bounded-length prefix of the input. Moreover, TLTL has a *qualitative semantics* in addition to the standard Boolean semantics of LTL: the denotation of a TLTL formula maps a sample path of the environment MDP to a real number that indicates the degree of satisfaction for the TLTL formula. Thus, I classified this approach as using an LTL-like objective.

Giacomo et al. [24] introduced *Restraining Bolts*. A Restraining Bolts specification is a set of pairs (ϕ_i, r_i), where each ϕ_i is an LTLf/LDLf formula, and r_i is a scalar reward. An LTLf formula is visuaully similar to an LTL formula; however, it is interpreted over finite-length words instead of infinite-length words. LDLf is an extension of LTLf and is also interpreted over finite-length words. [7] Given an environment MDP, the approach checks each finite length prefix of a sample path of the MDP against each ϕ_i, and if a prefix satisfies ϕ_i, the approach gives the corresponding reward r_i to the agent. The objective in Giacomo et al. [24] is to maximize the discounted cumulative rewards produced by the Restraining Bolts specification. To the best of my knowledge, this objective is not equivalent to maximizing the satisfaction of an LTL formula. Nonetheless, a Restraining Bolts specification can be seen as an LTL-like specification for its use of temporal operators. Therefore, I classified this approach as using an LTL-like objective.

Camacho et al. [23] introduced *reward machine*. A reward machine specification is a deterministic finite automaton equipped with a reward for each transition. The objective in Camacho et al. [23] is to maximize the discounted cumulative rewards produced by the

[7] LDLf is more expressive than LTLf [30]. In particular, LTLf is equally expressive as the star-free subset of regular languages while LDLf is equally expressive as the full set of regular languages.

reward machine specification. Camacho et al. [23] showed that LTL objectives formulas in the Guarantee or Safety class are reducible to reward machine objectives without discount factors. However, since the approach maximizes discounted cumulative rewards in practice, it does not directly optimize for the LTL objectives in the Guarantee or Safety classes.[8] Therefore, I classified this approach as using an LTL-like objective.

4.6 Concurrent Work

Concurrent to this work, Alur et al. [27] developed a framework to study reductions between reinforcement-learning task specifications. They looked at various task specifications, including cumulative discounted rewards, infinite-horizon average-rewards, reachability, safety, and LTL. They thoroughly review previous work concerning reinforcement learning for LTL objectives, which I also cite. Moreover, Alur et al. [27, Theorem 8] states a seemingly similar result as the forward direction of Theorem 4.3.4:

> There does not exist a PAC-MDP algorithm for the class of safety specifications.

Despite the parallels, there is a crucial difference and two nuances between my work and theirs, which I discuss in detail below.

Firstly and most importantly, their theorem is equivalent to "there exists a safety specification that is not PAC-learnable.", whereas Theorem 4.3.4 works pointwise for each LTL formula, asserting "all non-finitary specifications are not PAC-learnable." The proof of their theorem gives one safety specification and shows that it is not PAC-learnable.[9] On the other hand, the proof of the forward direction of Theorem 4.3.4 constructs a counterexample for each non-finitary formula. This point is crucial since it allows me to precisely carve out the PAC-learnable subset, namely the finitary formulas, from the LTL hierarchy. Secondly, their notion of sample complexity is slightly different from mine. In particular, they formulated a reinforcement learning algorithm as an iterative algorithm. At each step, the iterative algorithm outputs a policy π^n. Then, their notion of sample complexity is the total number of non-ϵ-optimal policies produced during an infinitely long run of the learning algorithm: $\left|\{n \mid V^{\pi^n}_{\mathcal{M},\xi} < V^{\pi^*}_{\mathcal{M},\xi} - \epsilon\}\right|$. On the other hand, my notion of sample complexity is the number of samples required until the learning algorithm outputs ϵ-optimal policies. However,

[8] By their reduction, as the discount factor approach 1 in the limit, the learned policy for the reward machine becomes the optimal policy for given guarantee or safety LTL objective. Therefore, Camacho et al. [23] can also be classified as using an LTL surrogate objectives (for the subset of guarantee and safety objectives. Nonetheless, I classified this approach to the general LTL-like objectives category because reward machine objectives are more general than LTL objectives.

[9] Their result is similar to what I showed in Section 4.3.3.2, where I consider the particular guarantee formula $\mathsf{F}\, g_0$ and show that it is not PAC-learnable.

this difference is orthogonal to the core issue caused by infinite-horizon LTL formulas. In particular, I can adapt my theorem and proof to use their notion of sample complexity.

Thirdly, their definition of safety specification is equivalent to a strict subset of the safety class in the LTL hierarchy that I consider. In particular, their safety specification is equivalent to LTL formulas of the form $\mathsf{G}\,(a_1 \vee a_2 \vee \cdots \vee a_n)$, where each $a_i \in \Pi$ is an atomic proposition, with $n = 0$ degenerating the specification to T (the constant true).

Lastly, they consider only reinforcement-learning algorithms, whereas I consider the slightly more general planning-with-generative-model algorithms. I believe their theorem and proof can be modified to accommodate my more general algorithm definition.

4.7 Proofs

4.7.1 Proof of Lemma 4.3.5

To the end of proving Lemma 4.3.5, I first observe the following proposition:

Proposition 4.7.1. *For any non-Markovian policy π, the satisfaction probabilities for $\mathcal{M}_1$ and $\mathcal{M}_2$ sum to one:*

$$V^{\pi}_{\mathcal{M}_1,\xi^{g_0}} + V^{\pi}_{\mathcal{M}_2,\xi^{g_0}} = 1.$$

I give a proof of Proposition 4.7.1 in Section 4.7.2.

Proof of Lemma 4.3.5. Note that the optimal satisfaction probabilities in both $\mathcal{M}_1$ and $\mathcal{M}_2$ is one, that is, $V^{\pi^*}_{\mathcal{M}_i,\xi^{g_0}} = 1$. This is because the policy that always chooses a_i in $\mathcal{M}_i$ guarantees visitation to the state g_0. Therefore, a corollary of Proposition 4.7.1 is that for any policy π and any $\epsilon < \frac{1}{2}$, the policy π can only be ϵ-optimal in one of $\mathcal{M}_1$ and $\mathcal{M}_2$. Specifically, I have:

$$\mathbb{1}\{\Big(V^{\pi}_{\mathcal{M}_1,\xi^{g_0}} \geq V^{\pi^*}_{\mathcal{M}_1,\xi^{g_0}} - \epsilon\Big)\} + \mathbb{1}\{\Big(V^{\pi}_{\mathcal{M}_2,\xi^{g_0}} \geq V^{\pi^*}_{\mathcal{M}_2,\xi^{g_0}} - \epsilon\Big)\} \leq 1. \tag{4.5}$$

Consider a specific sequence of transitions T of length N sampled from either $\mathcal{M}_1$ or $\mathcal{M}_2$. If $n(T) = 0$, the probability of observing T in $\mathcal{M}_1$ equals to the probability of observing T in $\mathcal{M}_2$, that is:

$$\begin{aligned}&\mathsf{P}_{\mathsf{T}\sim\langle\mathcal{M}_1,\mathcal{A}^{\mathrm{S}}\rangle_N}(\,\mathsf{T} = T \mid n(T) = 0\,)\\ &= \mathsf{P}_{\mathsf{T}\sim\langle\mathcal{M}_2,\mathcal{A}^{\mathrm{S}}\rangle_N}(\,\mathsf{T} = T \mid n(T) = 0\,).\end{aligned}$$

This is because the only differences between $\mathcal{M}_1$ and $\mathcal{M}_2$ are the transitions $q_l \to g_0$ and $q_l \to h_0$, and conditioning on $n(T) = 0$ effectively eliminates these differences.

Therefore, I can write the sum of ζ_1 and ζ_2 as:

$$\zeta_1 + \zeta_2 = \sum_{\forall T} \mathsf{P}_{\mathsf{T}}(\, \mathsf{T} = T \mid n(T) = 0\,) \times \left\{ \mathbb{1}\{\left(V^{\mathcal{A}^{\mathrm{L}}(T)}_{\mathcal{M}_1,\xi^{g_0}} \geq V^{\pi^*}_{\mathcal{M}_1,\xi^{g_0}} - \epsilon\right)\} + \mathbb{1}\{\left(V^{\mathcal{A}^{\mathrm{L}}(T)}_{\mathcal{M}_2,\xi^{g_0}} \geq V^{\pi^*}_{\mathcal{M}_2,\xi^{g_0}} - \epsilon\right)\}\right\}.$$

Plugging in Equation (4.5), I get

$$\zeta_1 + \zeta_2 \leq 1.$$

This then implies that $\min(\zeta_1, \zeta_2) \leq \frac{1}{2}$. □

4.7.2 Proof of Proposition 4.7.1

Proof. I first focus on $\mathcal{M}_1$. Consider an infinite run $\tau = (s_0, a_0, s_1, a_1, \dots)$ of the policy π on $\mathcal{M}_1$. Let $\tau[:i]$ denote the partial history up to state s_i; let w denote all the states $(s_0, s_1, \dots)$ in τ. Let E_i denote the event that the visited state at step i is either g_0 or h_0: $w[i] \in \{g_0, h_0\}$. I have:

$$\begin{aligned} V^{\pi}_{\mathcal{M}_1,\xi^{g_0}} &= \mathsf{P}(\mathcal{L}(w) \vDash \mathsf{F}\, g_0) \\ &= \sum_{i=1}^{\infty} \mathsf{P}(\, \mathcal{L}(w) \vDash \mathsf{F}\, g_0 \mid E_i\,) \cdot \mathsf{P}(E_i) \end{aligned}$$

Given that E_i happens, the previous state $w[i-1]$ must be q_l. Then, the probability of satisfying the formula given the event E_i is the probability of the learned policy choosing a_1 from the state q_l after observing the partial history $\tau[i-1]$:

$$V^{\pi}_{\mathcal{M}_1,\xi^{g_0}} = \sum_{i=1}^{\infty} \mathsf{P}(\, \pi\,(\tau[:i-1]) = a_1 \mid E_i\,) \cdot \mathsf{P}(E_i). \tag{4.6}$$

Symmetrically for $\mathcal{M}_2$ I then have:

$$V^{\pi}_{\mathcal{M}_2,\xi^{g_0}} = \sum_{i=1}^{\infty} \mathsf{P}(\, \pi\,(\tau[:i-1]) = a_2 \mid E_i\,) \cdot \mathsf{P}(E_i). \tag{4.7}$$

For any policy and any given partial history, the probability of choosing a_1 or a_2 must sum to 1, that is:

$$\mathsf{P}(\, \pi\,(\tau[:i-1]) = a_1 \mid E_i\,) + \mathsf{P}(\, \pi\,(\tau[:i-1]) = a_2 \mid E_i\,) = 1$$

Therefore, I may add Equation (4.6) and Equation (4.7) to get:

$$V^{\pi}_{\mathcal{M}_1,\xi^{g_0}} + V^{\pi}_{\mathcal{M}_2,\xi^{g_0}} = \sum_{i=1}^{\infty} 1 \cdot \mathsf{P}(E_i).$$

Finally, since the event E_i must happen for some finitary i with probability 1 (i.e., either g_0 or h_0 must be reached eventually with probability 1), the expression on the right of the equation sums to 1. □

4.7.3 Complete Proof of Lemma 4.3.6

Proof. First, consider $\mathcal{M}_1$. I will derive a lower bound for N. I begin by asserting that the inequality of Equation (4.2) holds true for a reinforcement-learning algorithm $\mathcal{A} = (\mathcal{A}^{\mathrm{S}}, \mathcal{A}^{\mathrm{L}})$. That is:

$$\mathsf{P}_{\mathsf{T}}\left(V^{\mathcal{A}^{\mathrm{L}}(T)}_{\mathcal{M}_1,\xi^{g_0}} \geq V^{\pi^*}_{\mathcal{M}_1,\xi^{g_0}} - \epsilon\right) \geq 1 - \delta.$$

I expand the left-hand side by conditioning on $n(T) = 0$:

$$\begin{aligned}\mathsf{P}_{\mathsf{T}}\left(V^{\mathcal{A}^{\mathrm{L}}(T)}_{\mathcal{M}_1,\xi^{g_0}} \geq V^{\pi^*}_{\mathcal{M}_1,\xi^{g_0}} - \epsilon \,\middle|\, n\left(T\right) = 0\right)\mathsf{P}_{\mathsf{T}}(n\left(T\right) = 0) + \\ \mathsf{P}_{\mathsf{T}}\left(V^{\mathcal{A}^{\mathrm{L}}(T)}_{\mathcal{M}_1,\xi^{g_0}} \geq V^{\pi^*}_{\mathcal{M}_1,\xi^{g_0}} - \epsilon \,\middle|\, n\left(T\right) > 0\right)\left(1 - \mathsf{P}_{\mathsf{T}}(n\left(T\right) = 0)\right) \\ \geq 1 - \delta.\end{aligned}$$

Since $\mathsf{P}_{\mathsf{T}}\left(V^{\mathcal{A}^{\mathrm{L}}(T)}_{\mathcal{M}_1,\xi^{g_0}} \geq 1 - \epsilon \,\middle|\, n\left(T\right) > 0\right) \leq 1$, I may relax the inequality to:

$$(1 - \zeta_1)\mathsf{P}_{\mathsf{T}}(n\left(T\right) = 0) \leq \delta,$$

where I also plugged in my definition of ζ_i (see Lemma 4.3.5). This relaxation optimistically assumes that a reinforcement-learning algorithm can learn an ϵ-optimal policy by observing at least one transition to g_0 or h_0.

Since there are at most N transitions initiating from the state q_l, and $n(T) = 0$ only occurs when all those transitions end up in q_k, I have $\mathsf{P}_{\mathsf{T}}(n\left(T\right) = 0) \geq (1-p)^N$. Incorporating this into the inequality I have:

$$(1 - \zeta_1)\left(1 - p\right)^N \leq \delta.$$

Symmetrically, for $\mathcal{M}_2$ I have:

$$(1 - \zeta_2)\left(1 - p\right)^N \leq \delta.$$

Since both inequalities need to hold, I combine them to choose the tighter inequality:

$$(1 - \min(\zeta_1, \zeta_2))(1-p)^N \leq \delta.$$

By applying Lemma 4.3.5, I remove the inequality's dependence on ζ_i, and get the desired lower bound of

$$N \geq \frac{\log(2\delta)}{\log(1-p)},$$

which completes the proof of Lemma 4.3.6. □

4.7.4 Uncommittable Words for non-*Finitary* Formulas

In this section, I prove the following lemma:

Lemma 4.7.2. *Any LTL formula $\phi \notin$ Guarantee has an uncommittable accepting word. Any LTL formula $\phi \notin$ Safety has an uncommittable rejecting word.*

4.7.4.1 Preliminaries

I will review some preliminaries to prepare for my proof of Lemma 4.7.2.

I will use an automaton-based argument for my proof of Lemma 4.7.2. To that end, I recall the following definitions for automatons.

Deterministic Finite Automaton A deterministic finite automaton (DFA) is a tuple $(S, A, P, s_0, s_{\text{acc}})$, where (S, A, P, s_0) is a deterministic MDP (i.e., P degenerated to a deterministic function $(S \times A) \to S$), and $s_{\text{acc}} \in S$ is an accepting state.

Deterministic Rabin Automaton A deterministic Rabin automaton (DRA) is a tuple (S, Π, T, s_0, Acc), where

- S is a finite set of states.
- Π is the atomic propositions of ϕ.
- T is a transition function $(S \times 2^{\Pi}) \to S$.
- $s_0 \in S$ is an initial state.
- Acc is a set of pairs of subsets of states $(B_i, G_i) \in (2^S)^2$.

An infinite-length word w over the atomic propositions Π is accepted by the DRA, if there exists a run of the DRA such that there exists a $(B_i, G_i) \in Acc$ where the run visits all states in B_i finitely many times and visits some state(s) in G_i infinitely many times.

For any LTL formula ϕ, one can always construct an equivalent DRA that accepts the same set of infinite-length words as ϕ [50].

4.7.4.2 Proof of Lemma 4.7.2 for $\phi \notin$ *Guarantee*

Given an LTL formula ϕ, I first construct its equivalent DRA $\mathcal{R} = (S, \Pi, T, s_0, Acc)$ [50].

A *path* in a DRA is a sequence of transitions in the DRA. A *cycle* in a DRA is a path that starts from some state and then returns to that state. A cycle is *accepting* if there exists a pair $(B_i, G_i) \in Acc$, such that the cycle does not visit states in B_i and visits some states in G_i. Conversely, a cycle is *rejecting* if it is not accepting. With the above definitions and to the end of proving Lemma 4.7.2 for the case of $\phi \notin$ *Guarantee*, I prove the following lemma.

Lemma 4.7.3. *For any LTL formula $\phi \notin$ Guarantee and its equivalent DRA $\mathcal{R}$, it must be the case that $\mathcal{R}$ contains an accepting cycle that is reachable from the initial state and there exists a path from a state in the accepting cycle to a rejecting cycle.*

Proof. Suppose, for the sake of contradiction, there does not exist an accepting cycle that 1. is reachable from the initial state and 2. has a path to a rejecting cycle in the equivalent DRA $\mathcal{R}$. Then there are two scenarios:

- $\mathcal{R}$ does not have any accepting cycle that is reachable from the initial state.
- All accepting cycles reachable from the initial state is not reachable to a rejecting cycle.

For the first scenario, $\mathcal{R}$ must not accept any infinite-length word. Therefore ϕ must be equal to F (i.e., the constant falsum). However, F is in the *Finitary* LTL class, which is a subset of *Guarantee*, so this is a contradiction.

For the second scenario, consider any infinite-length word w. Consider the induced infinite path $\mathcal{P} = (s_0, w[0], s_1, w[1], \dots)$ by w on the DRA starting from the initial state s_0.

If ϕ accepts the word w, the path $\mathcal{P}$ must reach some state in some accepting cycle.

Conversely, if ϕ rejects the word w, the path must not visit any state in any accepting cycle. This is because otherwise the path can no longer visit a rejecting cycle once it visits the accepting cycle, thereby causing the word to be accepted.

Therefore, ϕ accepts the word w as soon as the path $\mathcal{P}$ visits some state in some accepting cycle. This degenerates the DRA to a DFA, where the accepting states are all the states in the accepting cycles of the DRA. Then, an infinite-length word w is accepted by ϕ if and only if there exists a prefix of w that is accepted by the DFA.

By the property of the *Guarantee* class (see Section 4.2.3), for $\phi \in$ *Guarantee*, there exists a language of finite-length words, L, such that $w \vDash \phi$ if L accepts a prefix of w [43]. Since a DFA recognizes a regular language, the formula must be in the *Guarantee* LTL class. This is also a contradiction.

Therefore, there must exist an accepting cycle that is reachable from the initial state and has a path to a rejecting cycle in the equivalent DRA. This completes the proof of Lemma 4.7.3 □

I am now ready to give a construction of w_a, w_b, w_c and w_d that proves Lemma 4.7.2 for $\phi \notin$ *Guarantee*. Consider the equivalent DRA $\mathcal{R}$ of the LTL formula. By Lemma 4.7.3, $\mathcal{R}$ must contain an accepting cycle that is reachable from the initial state and has a path to a rejecting cycle. I can thus define the following paths and cycles:

- Let $\mathcal{P}_a$ be the path from the initial state to the accepting cycle.
- Let $\mathcal{P}_b$ be the accepting cycle.
- Let $\mathcal{P}_c$ be a path from the last state in the accepting cycle to the rejecting cycle.
- Let $\mathcal{P}_d$ be the rejecting cycle.

For a path $\mathcal{P} = (s_i, w[i], \dots s_j, w[j], s_{j+1})$, let $w(\mathcal{P})$ denote the finite-length word consisting only of the characters in between every other state (i.e., each character is a tuple of truth values of the atomic propositions): $w(\mathcal{P}) = w[i] \dots w[j]$. Consider the assignments of $w_a = w(\mathcal{P}_a)$, $w_b = w(\mathcal{P}_b)$, $w_c = w(\mathcal{P}_c)$ and $w_d = w(\mathcal{P}_d)$. Notice that:

- The formula ϕ accepts the infinite-length word $[w_a; w_b^\omega]$ because P^b is an accepting cycle.
- The formula ϕ rejects all infinite-length words $[w_a; w_b^i; w_c; w_d^\omega]$ for all $i \in \mathbb{N}$ because P^d is a rejecting cycle.

By Definition 4.3.7, the infinite-length word $[w_a; w_b^\omega]$ is an uncommittable accepting word. This construction proves Lemma 4.7.2 for $\phi \notin$ *Guarantee*.

4.7.4.3 Proof of Lemma 4.7.2 for $\phi \notin$ *Safety*

The proof for $\phi \notin$ *Safety* is symmetrical to $\phi \notin$ *Guarantee*, which I give below for completeness.

Given an LTL formula ϕ, I again first construct its equivalent DRA $\mathcal{R} = (S, \Pi, T, s_0, Acc)$.

To the end of proving Lemma 4.7.2 for the case of $\phi \notin$ *Safety*, I state and prove the following lemma.

Lemma 4.7.4. *For any LTL formula $\phi \notin$ Safety and its equivalent DRA $\mathcal{R}$, it must be the case that $\mathcal{R}$ contains a rejecting cycle that is reachable from the initial state and has a path from any state in the rejecting cycle to an accepting cycle.*

Proof. Suppose, for the sake of contradiction, there does not exist a rejecting cycle that 1. is reachable from the initial state and 2. has a path to an accepting cycle in the equivalent DRA $\mathcal{R}$. Then there are two scenarios:

- $\mathcal{R}$ does not have any rejecting cycle that is reachable from the initial state.
- All rejecting cycles reachable from the initial state is not reachable to an accepting cycle.

For the first scenario, $\mathcal{R}$ must not reject any infinite-length word. Therefore ϕ must be equal to T (i.e., the constant truth). However, T is in the *Finitary* LTL class, which is a subset of *Safety*, so this is a contradiction.

For the second scenario, consider any infinite-length word w. Consider the induced infinite path $\mathcal{P} = (s_0, w[0], s_1, w[1], \dots)$ by w on the DRA starting from the initial state s_0.

If ϕ rejects the word w, the path $\mathcal{P}$ must reach some state in some rejecting cycle.

Conversely, if ϕ accepts w, the path must not visit any state in any rejecting cycle. This is because otherwise the path can no longer visit a accepting cycle once it visits the rejecting cycle, thereby causing the word to be rejected.

Therefore, ϕ rejects the word w as soon as the path $\mathcal{P}$ visits some state in some rejecting cycle. I can again construct a DFA based on the DRA by letting the accepting states be all the states except those in a rejecting cycle of the DRA. By this construction, ϕ accepts an infinite-length word w if and only if the DFA accepts all finite-length prefixes of w.

By the property of the *Safety* class (see Section 4.2.3), for $\phi \in$ *Safety*, there exists a language of finite-length words, L, such that $w \vDash \phi$ if L accepts all prefixes of w [43]. Since a DFA recognizes a regular language, the formula must be in the *Safety* LTL class. This is also a contradiction. Therefore, there must exist a rejecting cycle that is reachable from the initial state and has a path to an accepting cycle in the equivalent DRA. This completes the proof of Lemma 4.7.4 □

I am now ready to give a construction of w_a, w_b, w_c and w_d that proves Lemma 4.7.2 for $\phi \notin$ *Safety*. Consider the equivalent DRA $\mathcal{R}$ of the LTL formula. By Lemma 4.7.4, $\mathcal{R}$ must contain a rejecting cycle that is reachable from the initial state and has a path to an accepting cycle. I can thus define the following paths and cycles:

- Let $\mathcal{P}_a$ be the path from the initial state to the rejecting cycle.
- Let $\mathcal{P}_b$ be the rejecting cycle.

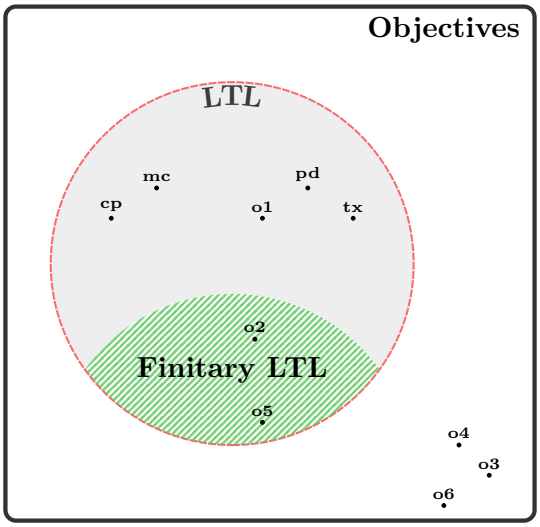


Figure 4.5: Landscape of objectives' learnability up to the current chapter. Dashed circle: LTL objectives; Gray area: infinite-horizon LTL objectives, not PAC-learnable. Green slanted area: finitary LTL objectives, PAC-learnable. Empty area: objectives not expressible by LTL formulas.

- Let $\mathcal{P}_c$ be a path from the last state in the rejecting cycle to the accepting cycle.
- Let $\mathcal{P}_d$ be the accepting cycle.

Consider the assignments of $w_a = w(\mathcal{P}_a)$, $w_b = w(\mathcal{P}_b)$, $w_c = w(\mathcal{P}_c)$ and $w_d = w(\mathcal{P}_d)$. Then we have the following facts:

- The formula ϕ rejects the infinite-length word $[w_a; w_b^\omega]$ because P^b is a rejecting cycle.
- The formula ϕ accepts all infinite-length words $[w_a; w_b^i; w_c; w_d^\omega]$ for all $i \in \mathbb{N}$ because P^d is an accepting cycle.

By Definition 4.3.7, the infinite-length word $[w_a; w_b^\omega]$ is an uncommittable rejecting word. This construction proves Lemma 4.7.2 for $\phi \notin \mathit{Safety}$.

4.8 Chapter Summary

Figure 4.5 summarizes the landscape of example objectives' learnability up to the current chapter. The objectives **o1**, **o2**, **o5**, **cp**, **mc**, **pd**, **tx** are expressible as LTL objectives. Among them, **o2** and **o5** are in the finitary LTL class, and reinforcement learning exists that learns near-optimal policies for them with high probability. On the other hand, **o1**, **cp**, **mc**, **pd**, **tx** are not in the finitary LTL class. For these objectives, the core result in

this chapter implies that no reinforcement learning can learn near-optimal policies with high probability. Particularly for standard benchmark reinforcement-learning objectives such as the cart-pole (**cp**), mountain-car (**mc**), pendulum (**pd**), and taxi (**tx**) objectives, albeit many reinforcement-learning algorithms empirically produce policies that perform well in the benchmark environment for each of these objectives, our core results suggests that no reinforcement learning algorithm can produce near-optimal policies with a high probability guarantee in an arbitrary unrestricted environment. For example, if the cart-pole happens on a planet with unknown laws of physics, no reinforcement learning algorithm can learn near-optimal policies with high probability. For other objectives, such as **o4** and **o3**, they are not expressible by LTL formulas — I discuss these objectives in the following chapters.

In this chapter, I have formally proved that infinite-horizon LTL objectives in reinforcement learning cannot be learned in unrestricted environments. By inspecting the core result, I have identified various possible directions forward for future research. My work resolves the apparent lack of a formal treatment of this fundamental limitation of infinite-horizon objectives, helps increase the community's awareness of this problem, and will help organize the community's efforts in reinforcement learning with LTL objectives.

Chapter 5

On the Learnability of Computable Objectives

In Chapter 4, I have shown that only finitary LTL objectives — those that require inspecting only a finite number of trajectory steps to determine if the trajectory satisfies the objective — are PAC-learnable.

5.1 Overview

A natural question is if we may generalize this result to other objectives: What other kinds of objectives are PAC-learnable?

Recall the gridworld example objectives in Chapter 1, repeated at Table 5.1 for convenience. In particular, consider the objectives **o2** and **o3**. The objective **o2** requires reaching the goal but avoiding water within a fixed number of steps. In Chapter 4, I have established that this objective is PAC-learnable. On the other hand, the objective **o3** is similar to **o2**: instead of a fixed number of steps, the objective **o3** requires reaching the goal within a geometrically distributed number of steps. The geometric distribution has a known finite

Example Gridworld Objectives	
o1	Reach goal without stepping on water
o2	Do **o1** within $n = 15$ steps
o3	Do **o1** within $n \sim \text{Geom}(\frac{1}{15})$ steps
o4	Do **o1**, then retrace the steps back
o5	Do **o4** within $n = 30$ steps
o6	Repeat **o4** forever

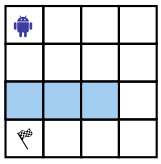


Table 5.1: Example gridworld objectives repeated from Table 1.2.

expected value — so intuitively, we require the agent to perform the task within a finite number of steps on average. Therefore, we expect that **o3** is also PAC-learnable because it is similar to **o2** in that it does not require inspecting an infinite number of steps of the trajectory. To see this later fact, recall that the geometric distribution assigns a zero probability to the event that requires an infinite number of steps. Despite the similarity, **o3** is not an LTL objective: An LTL objective is a deterministic Boolean function over trajectories, but **o3** is a function that maps from trajectories to real numbers in $[0, 1]$, where the value is the probability value of satisfying the task within the geometrically distributed number of steps. Hence, the results in Chapter 4 do not directly apply to the objective **o3**.

In this chapter, I will show that the objective **o3** is PAC-learnable. The conventional approach to proving the PAC-learnability of an objective is to design a reinforcement learning algorithm that learns a near-optimal policy for the objective. However, this approach requires reasoning about the environment, the policy, and the algorithm. Instead, I establish a general condition that any continuous and computable objective is PAC-learnable. In particular, I prove that any continuous objective is PAC-learnable in the sample sense, and any computable objective is PAC-learnable in the computational sense.

This condition simplifies proving PAC-learnability of objectives: It avoids reasoning about the environment, the policy, and the algorithm but only requires reasoning about the objective itself. Further, this condition provides an approach to learning a near-optimal policy for any computable objective with a PAC guarantee.

In the following section, I will use objective **o3** as an example to illustrate this approach.

5.1.1 Example

Consider the objective **o3** "reach the goal and avoid water within a geometrically distributed number of steps." As mentioned above, this objective is not an LTL objective; however, it also does not require inspecting an infinite number of trajectory steps to determine if the trajectory satisfies the objective. This property of "not requiring inspecting an infinite number of steps of a trajectory" is formally capturable by the notion of *computability* in computability theory — in particular, Type-2 computability theory, a generalization of computability theory to infinite objects.

A formal connection requires formalizing the objective as a function from trajectories to real numbers. It turns out that this objective falls into a class of known objective functions in the literature, called *geometric linear temporal logic* (GLTL) objectives [21]. The main chapter will show that this class of objectives is computable and use the core condition to prove that it is PAC-learnable. However, for this overview and to avoid introducing much

technical machinery early on, I here formally define this objective in the following way: First, draw a geometrically distributed number of steps T from the geometric distribution with the parameter $p = \frac{1}{15}$. Then, consider the finitary LTL objective that requires reaching the goal and avoiding water within T steps. The objective **o3** is formally defined as maximizing the probability of satisfying this finitary LTL objective.

Next, I will use this example objective to illustrate the core condition.

5.1.2 Continuity and Computability

Continuity and computability are two key properties of objectives that I will use to establish the core condition. They are also related to each other: A classic result in computability theory states that any computable objective is continuous [51]. So, computability is a stronger condition than continuity. In the following overview, I will first define these two properties informally and illustrate the definitions with the example objective **o3**, which formal treatment will follow in the main chapter.

Continuity A function is continuous if small variations in the input result in small variations in the output. Since objectives are functions from trajectories to real numbers, a continuous objective maps from trajectories to real numbers, and small variations in the input trajectory result in small variations in the output real value. In particular, by a standard metric space definition over trajectories, which I cover in the main chapter, two trajectories has a small variation of ϵ — formally speaking, ϵ-close — if they share a common prefix of length $T = -\log_2(\epsilon)$. In words, the longer the common prefix of two trajectories, the closer they are to each other in this metric space.

Our example objective **o3** is, in fact, continuous under this definition. Consider two trajectories that share a common prefix of length T. Then, the value of this objective on the two trajectories differs by at most $(1-p)^T$. To see this, recall that the probability of the objective requesting a horizon of more than T steps is exactly $(1-p)^T$. For horizons less than T, the two trajectories are identical, so the probability of satisfying the objective on the two trajectories is the same. The two trajectories differ for horizons larger than T, but the probability of that happening is at most $(1-p)^T$. Therefore, for any two trajectories that share a common prefix T (i.e., they are ϵ-close, where $\epsilon = 2^{-T}$), the probability of satisfying the objective on the two trajectories differs by at most $(1-p)^T$. In particular, since $1-p$ is less than one, this exponential dependence on T implies that the closer the two trajectories are to each other, the closer the objective values on the two trajectories are to each other. Therefore, the objective **o3** is continuous.

```python
def reach_no_water_geom(traj: Iterator[State], T: int) -> Rational:
 return all(traj[:T] is not water) and any(traj[:T] is goal)
```

Listing 5.1: Psuedocode of the objective **o3** up to a finite number of steps T.

Computability A computable objective is one that a computer program can compute. However, since the objective is a function from trajectories to real numbers, and each trajectory is an infinite sequence of states and actions, and real numbers are uncountable, both the domain and codomain of the objective are *infinite objects* that cannot be stored in a finite amount of memory or computed in a finite amount of time. Therefore, the notion of computability requires a generalization of computability theory to infinite objects, provided by a branch of computability theory called Type-2 computability theory [51].

In Type-2 computability theory, an objective is computable if a computer program computes the function in the streaming sense. Specifically, the program reads from an infinite stream of states and writes to an infinite stream of rational outputs. The program runs forever, reading infinitely many inputs and writing infinitely many outputs. In the limit, the output rational numbers converge to the real-valued output of the objective function. Equivalently, the program computes the objective function if it computes an arbitrary approximation to the objective function on finite prefixes of the trajectory, that is: For any finite prefix of the trajectory, the program computes an output that is within the specified precision of the true output, and the number of steps required to obtain the output is computable from the precision. Computability is a stronger condition than continuity: A computable objective is always continuous.

In our example, the objective **o3** is both continuous and computable. Specifically, for an arbitrary precision ϵ, we can compute the number of steps T required to obtain an output within ϵ of the true output. Then, we can compute the finitary LTL objective that requires reaching the goal and avoiding water within T steps. To see this, consider the computation in Listing 5.1 that reads from a stream of states and actions, and checks if the trajectory satisfies "reach the goal and avoid water within T steps." If we run this computation for some input T and compare it to the true output of the objective, we will find that the output differs by at most $(1-p)^T$ to the true output because the probability of requiring more than T steps is exactly $(1-p)^T$, and even though our computation is potentially wrong for any case where the true objective requires more than T steps, the probability of that happening at most $(1-p)^T$, and since the objective's value is between 0 and 1, the error is at most $(1-p)^T$. Therefore, for any precision ϵ, we can choose $T = \log_{(1-p)}(\epsilon)$ such that the error

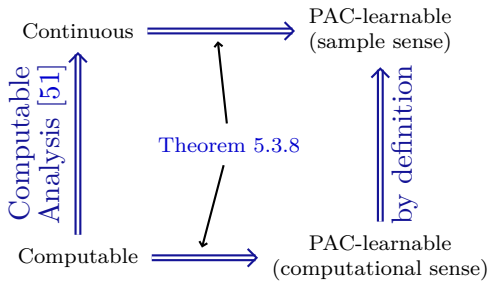


Figure 5.1: Conditions for PAC-learnability

is less than ϵ. By running the computation in Listing 5.1 for T steps, we obtain an output within ϵ of the true output.

5.1.3 PAC-learnability

I next show that the objective **o3** is PAC-learnable.

Recall from Chapter 3 that PAC-learnability requires learning a near-optimal policy with high probability. Since the objective is computable, we can compute an arbitrary approximation to the objective function. Thus, we can use the computation in Listing 5.1 as an approximation to the objective function. By learning a policy for this approximation, we can ensure that this learned policy is also near-optimal for the true objective. In particular, suppose we would like to learn a policy that is ϵ-optimal with probability $1 - \delta$. Then, we choose $T = \log_{(1-p)}(\frac{\epsilon}{2})$ such that the error of the approximate computable objective is less than $\frac{\epsilon}{2}$. We then follow the same approach as in Chapter 4 to reduce the problem of policy learning for the computable approximate objective, equivalent to a finitary LTL objective, to the problem of policy learning for a finite-horizon cumulative rewards objective. Same as in Chapter 4, we can use a reinforcement learning algorithm such as the ORLC algorithm provided by Dann et al. [38] for finite-horizon cumulative reward objectives to learn a near-optimal behavior for this reward objective up to precision $\frac{\epsilon}{2}$ with probability $1 - \delta$. By combining the two errors, each less than $\frac{\epsilon}{2}$, we obtain tha a learned policy through such an approach is ϵ-optimal for the original objective **o3** with probability $1 - \delta$.

The overall approach above applies not only to the example objective **o3**, but to any continuous or computable objective. In the following sections, I will show that a continuous objective is PAC-learnable in the sample sense and a computable objective is PAC-learnable in the computational sense. Figure 5.1 summarizes these conditions, the relationship between continuity and computability as discussed above, and the relationship between PAC-learnability in the sample and computational sense. In the following sections, I formally establish the arrows in Figure 5.1. In Section 5.4, I will apply these conditions to three

classes of objectives and show that they are PAC-learnable.

5.2 Type-2 Computability Theory

Computabilty theory is a foundational concept in computer science, and it provides a theoretical framework for understanding the limits of computation. In this chapter, I reveal a connection between reinforcement learning objectives and computability, and this section provides an overview of the key concepts in Type-2 computability theory, focusing on computability over infinite objects, such as infinite sequences and real numbers. This section's material closely follows the exposition in Weihrauch [51], which comprehensively treats Type-2 computability theory.

5.2.1 Ordinary Computability

Computability theory studies the notion of computable functions, which are functions that a computer program can compute. Before introducing Type-2 computability theory, which extends the notion of computability to infinite objects, I first review the standard notion of computability over finite objects.

I use the widely accepted notion of computability over finite objects such as $\mathbb{N}, \mathbb{Q}, \Sigma^*$: A function mapping between these objects is computable iff a program in a Turing-complete programming language supporting types of these objects computes that function. Type-2 computability theory extends this notion to functions over uncountable objects, such as infinite sequences and real numbers, which will be relevant in the thesis. However, to provide a soft introduction to Type-2 computability theory, I first briefly review the standard notion of computability over finite objects based on Turing machines.

The ordinary notion of computability, or Type-1 computability, considers functions over finite objects, such as integers, rationals, or finite-length words. In the following overview, I assume familiarity with the standard definition of an ordinary Turing machine and its operation on finite-length words.

Let Σ be a finite set of symbols called an *alphabet*, and let Σ^* denote the set of all finite-length words over Σ. A function mapping between finite-length words $f\colon \Sigma^* \to \Sigma^*$ is *computable* if there exists a Turing machine that computes f, meaning the Turing machine halts on input x and outputs $f(x)$ for all $x \in \Sigma^*$.

Notations Computability over other countable sets, such as integers or rationals, is defined in terms of naming elements of these finite-length words using a notation function. A notation

for a set M is a partial surjective function $\nu \subseteq \Sigma^* \to M$.

Computability under Notations For a single argument function $f\colon M \to N$, the function is computable under notations ν_M and ν_N if there is a computable function $\phi\colon \Sigma^* \to \Sigma^*$ such that

$$f(\nu_M(x)) = \nu_N(\phi(x))$$

for all $x \in \Sigma^*$ such that $f(\nu_M(x))$ is defined. Intuitively, the definition says that a function is computable if there is a computable function that maps the notation of an element in M to the notation of the corresponding element in N. The Turing machine that computes f still transforms from finite-length words to finite-length words, and the user interprets the input and output as elements of M and N, respectively.

More broadly, a $k \geq 0$ argument function $f\colon (M_1 \times \ldots \times M_k) \to N$ is computable under notations $\nu_{M_1} \ldots \nu_{M_k}$ and ν_N if there is a computable function $\phi\colon (\Sigma^*)^k \to \Sigma^*$ such that

$$f(\nu_{M_1}(x_1), \ldots, \nu_{M_k}(x_k)) = \nu_N(\phi(x_1, \ldots, x_k))$$

for all $x_1, \ldots, x_k \in \Sigma^*$ such that $f(\nu_{M_1}(x_1), \ldots, \nu_{M_k}(x_k))$ is defined.

Pseudocode to Represent Computable Functions As mentioned, since computability over finite objects such as $\mathbb{N}, \mathbb{Q}, \Sigma^*$ is well-understood, in the thesis, I use the widely accepted notion of computability over finite objects to avoid explicitly reasoning about Turing machines and notations: A function mapping between these objects is computable if there is a program in a Turing-complete programming language that computes the function.

In particular, throughout the thesis, I use a Python-like pseudocode to represent computable functions. I assume standard support for finite objects such as integers, rationals, and finite-length words (lists) in this language. I also assume standard computable functions over these objects, such as arithmetic operations, indexing, and iteration. I intentionally avoid imprecise data types, such as floating-point numbers, to ensure the pseudocode matches the theoretical Type-1 computability definition.

5.2.2 Type-2 Computability

In this thesis, I discuss reinforcement-learning objectives as functions from trajectories, which are infinite-length words, to real numbers. Therefore, notions of computability on such uncountable objects become essential. Type-2 computability theory extends the classical notion of computability to functions over uncountable objects, such as infinite sequences or real numbers. This section provides a brief overview of Type-2 computability theory.

```
def finite_discounted_sum(w: List[Rational], gamma: Rational = 0.9) -> Rational:
  """Compute the discounted sum of the input sequence."""
  s: Rational = 0
  for i, x in enumerate(w):
    s += (gamma ** i) * x
  return s

def sum(w: List[Rational]) -> Rational:
  """Compute the sum of the input sequence."""
  return discounted_sum(w, 1)
```

Listing 5.2: Example pseudocode of a computable over finite objects.

Uncountable objects are infinite by their nature: Since there is no surjective function from such objects to the finite-length words Σ^*, they are not representable by a finite sequence of finite-length words, but only by an infinite sequence of finite-length words. Consequently, each object is not storable in a finite Turing machine's tape segment.

No physical device can store uncountable objects or perform infinite computations within finite time. However, the key insight of Type-2 computability is that a computer can operate on finite segments of inputs and compute corresponding finite segments of outputs. Thus, the notion of computability over infinite objects reduces to computability over sequences of finite objects representing these infinite objects in the limit.

5.2.2.1 Computability over Infinite-length Words

A Type-2 Turing machine is a Turing machine that operates on infinite-length words. Specifically, it has $k \in \mathbb{N}$ input tapes, a finite number of work tapes, and one output tape. The input tapes are read-only, and the output tape is write-only.

Unlike ordinary Turing machines, a Type-2 Turing machine never halts: It reads the infinite-length inputs from input tapes and runs the computation forever, writing the output to the output tape. We say a infinite-length word function $f\colon (\Sigma^\omega \times \cdots \times \Sigma^\omega) \to \Sigma^\omega$ is *computable* if there is a Type-2 Turing machine that, given $w_1, w_2, \ldots, w_k \in \Sigma^\omega$ as input, computes forever and writes $y = f(w_1, w_2, \ldots, w_k) \in \Sigma^\omega$ to the output tape.

An infinite-length word $x \in \Sigma^\omega$ is *computable* iff the constnat function with zero number of inputs $f() = x$ is computable.

5.2.2.2 Computability over General Sets

Just like computability over general countable sets is defined in terms of naming elements of finite-length words using a notation function, computability over general uncountable sets is

defined in terms of naming elements of infinite-length words using a *representation* function.

Representations A representation for a set M is a partial surjective function $\nu \subseteq \Sigma^\omega \to M$. A function $f\colon M \to N$ is computable under representations ν_M and ν_N if there is a computable function $\phi\colon \Sigma^\omega \to \Sigma^\omega$ such that

$$f(\nu_M(x)) = \nu_N(\phi(x))$$

for all $x \in \Sigma^\omega$ such that $f(\nu_M(x))$ is defined. Intuitively, the definition says that a function is computable if there is a computable function that maps the representation of an element in M to the representation of the corresponding element in N. The Type-2 Turing machine that computes f still transforms from infinite-length words to infinite-length words, and the user interprets the input and output as elements of M and N, respectively.

Computability under Representations More broadly, a $k \geq 0$ argument function $f\colon (M_1 \times \ldots \times M_k) \to N$ is computable under representations $\nu_{M_1} \ldots \nu_{M_k}$ and ν_N if there is a computable function $\phi\colon (\Sigma^\omega)^k \to \Sigma^\omega$ such that

$$f(\nu_{M_1}(x_1), \ldots, \nu_{M_k}(x_k)) = \nu_N(\phi(x_1, \ldots, x_k))$$

for all $x_1, \ldots, x_k \in \Sigma^\omega$ such that $f(\nu_{M_1}(x_1), \ldots, \nu_{M_k}(x_k))$ is defined.

5.2.2.3 Computability over Real Numbers

The real numbers $\mathbb{R}$ are uncountable, thus necessitating a representation function to define computability over them. In the thesis, I will use the most useful representation of real numbers, the *fast-converging Cauchy representation*. This representation is useful because practically relevant real functions are computable under this representation — I discuss some example functions after the definition.

The fast-converging Cauchy representation of a real number $x \in \mathbb{R}$ is a sequence of rational numbers $(q_n)_{n\in\mathbb{N}}$ such that

$$\forall n \in \mathbb{N}, \quad |x - q_n| \leq 2^{-n}. \tag{5.1}$$

Since rational numbers are countable, they can be represented by finite-length words on a tape, for example, by representing the numerator and denominator of the rational number in binary. Therefore, the fast-converging Cauchy representation of a real number maps an

```
def every_other(w: Iterator[A]) -> Iterator[A]:
  """Output every other element of the input infinite-length word."""
  for i, x in enumerate(w):
    if i % 2 == 0:
      yield x
```

Listing 5.3: Example pseudocode of a computable function over infinite-length words.

```
# type alias for the Cauchy representation of real numbers
Real = Iterator[Rational]

def discounted_sum(w: Iterator[Rational], gamma: Rational = 0.9) -> Real:
  """Compute the discounted sum of an infinite stream of rational numbers in [0, 1]."""
  for i in count(0):
    n = (log2floor(1 - gamma) - i) / log2ceil(gamma)
    yield finite_discounted_sum(list(islice(w, n)), gamma)
```

Listing 5.4: Example pseudocode of a computable over infinite objects.

infinite sequence of rationals — each encoded as a finite-length word — converging to that real number in the limit.

Since I will be concerned with only the fast-converging Cauchy representation of real numbers, by computable fundamental functions, I mean real functions that are computable under the fast-converging Cauchy representation. A real number $x \in \mathbb{R}$ is *computable* if the constant function with zero number of inputs $f() = x$ is computable.

As mentioned, many practical real functions are computable under the fast-converging Cauchy representation. For example, the arithmetic operations $+, -, \times, \div$, exponentiation x^y for $x, y \in \mathbb{R}$, the elementary functions $\exp, \log, \sin, \cos$, and many continuous piecewise functions such as $\min(x, y)$, $\max(x, y)$ and $|x|$ are computable in Type-2 computability theory. On the other hand, various common real functions are not computable. The following classic result states the inherent non-computability of the equality and comparison functions.

Theorem 5.2.1 (Non-computability of Equality and Comparison [51, Theorem 4.1.16])**.** *There is no representation of the reals such that the functions 1. $f(x, y) = \mathbb{1}\{x = y\}$ (equality) and 2. $f(x, y) = \mathbb{1}\{x \leq y\}$ (comparison) are computable in Type-2 computability theory.*

5.2.2.4 Psuedocode for Computable Functions over Infinite Objects

When proving the computability of a function over infinite objects, the Type-2 theoretical approach is tedious to work with since it requires specifying the function on a Turing machine.

Therefore, in the case of computability over finite objects, to prove the computability of a function over infinite objects, I avoid explicit reasoning about Type-2 Turing machines and representations, Instead, I use pseudocode to formulate an algorithm that takes in infinite streams of inputs $w_1 \dots w_k$ and outputs an infinite stream.

Specifically, I use a Python-based pseudocode to represent computable functions over infinite objects. In this language, I assume standard support for infinite objects, such as infinite-length words and real numbers, and standard computable functions over these objects, such as arithmetic operations, indexing, and iteration.

Listing 5.3 provides an example of a Type-2 computable function that operates on infinite-length words. Specifically, the function `every_other` takes an infinite-length word as input and outputs an infinite-length word containing every other input element.

I use the type alias `Iterator[A]` to represent infinite-length words of type `A`, corresponding to the mathematical notation A^ω. In this context, Python's generator mechanism serves as a natural representation for Type-2 computable functions, enabling functions to consume and produce infinite objects by consuming and yielding elements one at a time. In particular, each time a `yield x` statement is executed, the function emits an output `x` to the output stream, and by repeatedly yielding outputs, it produces an infinite output stream.

Additionally, I use standard Python utilities, such as `enumerate` in this example, to simplify the implementation. These utilities retain the semantics of their counterparts in Python's standard library. I provide a complete list of these utilities in Appendix A.

Listing 5.4 provides an example of a computable function mapping from an infinite stream of rational numbers (`w: Iterator[Rational]`) and a rational number γ (`gamma: Rational`) to a fast-converging Cauchy represented real number. The function computes the discounted sum of an infinite stream of rational numbers in $[0, 1]$. The type `Real`, as an alias to `Iterator[Rational]`, indicates that the variable of this type is a real number represented in the fast-converging Cauchy representation. However, this type hint is a notation of intent rather than a guarantee. To rigorously establish that a function, such as `discounted_sum`, is a Type-2 computable function that outputs a real number, it is necessary to prove that the sequence of yielded outputs explicitly satisfies the fast-converging Cauchy representation criteria. The proof must demonstrate that each rational number in the sequence converges to the corresponding real number in the limit, as specified by the condition described in Equation (5.1). For this example, the proof would involve showing that the sequence of outputs converges to the discounted sum of the input sequence in the limit. To see this, note that the maximum difference between the discounted sum in the limit and the i-th yielded

output is

$$\sum_{j=n(i)}^{\infty} \gamma^j = \frac{\gamma^{n(i)}}{1-\gamma} = \frac{\gamma^{\left(\frac{\lfloor \log_2(1-\gamma) \rfloor - i}{\lceil \log_2(\gamma) \rceil}\right)}}{1-\gamma} \leq \frac{\gamma^{\left(\frac{\log_2(1-\gamma)-i}{\log_2(\gamma)}\right)}}{1-\gamma} = 2^{-i}.$$

Therefore, the function `discounted_sum` is a Type-2 computable function that outputs the discounted sum of the input sequence in the fast-converging Cauchy representation.

5.2.2.5 Computability Implies Continuity

A classic result in computability theory is that all computable functions are continuous [51]. In particular, I recall the following classic theorem:

Theorem 5.2.2 (Computability implies Continuity, [51, Corollary 3.1.2])**.** *For $i = 0 \dots k$, let ν_i be representations of sets M_i, and τ_i be a topology on M_i. If each ν_i is* admissible *with respect to τ_i, then every computable function $f \colon (M_1 \times \ldots \times M_k) \to M_0$ under representations $\nu_1, \ldots, \nu_k, \nu_0$ is continuous with respect to the topologies $\tau_1, \ldots, \tau_k, \tau_0$.*

I only consider standard topologies in the spaces of interest: For example, the standard topology induced by d_Σ^ω for Σ^ω and the standard topology on $\mathbb{R}$ induced by the Euclidean metric. Further, all representations I consider are admissible concerning the standard topologies, such as the trivial identity representation of Σ^ω and the fast-converging Cauchy representation of real numbers. Therefore, the precise definition of admissibility is only a technicality that I omit here. For a precise definition, see Weihrauch [51, Chapter 3]. In summary, all computable functions are continuous with respect to the standard topologies of the spaces of interest.

5.3 Condition for PAC-Learnability

This section presents my main result: sufficient conditions for an objective's learnability. The first two subsections analyze learnability in the information-theoretic setting. Specifically, I show that an objective given as an oracle is PAC-learnable if it is uniformly continuous. The next two subsections analyze learnability in the computation-theoretic setting. Specifically, using a standard result from computational analysis, I show that a computable objective is PAC-learnable. Section 5.10 complements my result by showing that my conditions are sufficient but not necessary.

5.3.1 Uniform Continuity

I first recall the following standard definition of a uniformly continuous function.

Definition 5.3.1 (Uniformly Continuous Function)**.** *A function $f\colon X \to Y$ with metric spaces (X, d_X) and (Y, d_Y) is uniformly continuous if, for any $\epsilon > 0$, there exists $\delta > 0$ so that f maps δ-close elements in the domain to ϵ-close elements in the image* [1] *:*

$$\forall \epsilon > 0. \exists \delta > 0. \forall x_1 \in X. \forall x_2 \in X :$$
$$d_X(x_1, x_2) \leq \delta \Rightarrow d_Y(f(x_1), f(x_2)) \leq \epsilon.$$

To specialize the above definition to an objective, I next note the metric space of the domain of an objective. An objective's domain is the set of infinite-length sequences X^ω, where $X = (S \times A)$ for an environment-specific objective and $X = F$ for an environment-generic objective. The domain forms a metric space by the standard distance function $d_{X^\omega}(w_1, w_2) = 2^{-L_{\text{prefix}}(w_1, w_2)}$, where $L_{\text{prefix}}(w_1, w_2)$ is the length of the longest common prefix of w_1 and w_2 [43]. I now specialize the definition of uniform continuity to objectives.

Definition 5.3.2 (Uniformly Continuous Objective)**.** *An objective (environment-specific or environment-generic) $f\colon X^\omega \to \mathbb{R}$ is uniformly continuous if, for any $\epsilon > 0$, there exists a finite horizon H so that the objective maps all infinite-length sequences sharing the same prefix of length H to ϵ-close values:*

$$\forall \epsilon > 0. \exists H \in \mathbb{N}. \forall w \in X^\omega. \forall w' \in X^\omega :$$
$$L_{prefix}(w, w') \geq H \Rightarrow |f(w) - f(w')| \leq \epsilon.$$

Note that since the domain of an objective is compact, the Heine–Cantor theorem guarantees that a continuous objective is also uniformly continuous. This paper only uses uniform continuity since it is more relevant to my theorem and proof. Nonetheless, theorems presented in the following section also hold for continuous objectives.

5.3.2 Continuity Implies PAC-learnability

Environment-specific Objectives. I give a sufficient condition for a learnable environment-specific objective:

Theorem 5.3.3. *An environment-specific objective κ is κ-PAC-learnable in the information-theoretic setting if it is uniformly continuous.*

I will prove the theorem by constructing a κ-PAC reinforcement-learning algorithm for any uniformly continuous κ. To that end, I reduce κ to a finite-horizon cumulative rewards

[1] Note that textbook definitions commonly use $<$ instead of $\leq$: my definition is equivalent. I use $\leq$ to match with the comparison operators in the PAC definitions.

problem; I then prove the theorem by invoking an existing PAC reinforcement-learning algorithm for finite-horizon cumulative rewards problems.

Proof of Theorem 5.3.3. For any $\epsilon' > 0$, since the objective κ is uniformly continuous, there exists a bound H such that all infinite-length sequences sharing a length-H prefix are mapped to ϵ'-close values.

For concreteness, let us pick any $\dot{s} \in S$ and any $\dot{a} \in A$. For each length-H sequence $u \in (S \times A)^H$, I pick the representative infinite-length sequence $[u; (\dot{s}, \dot{a})^\omega]$ that starts with the prefix u and ends in an infinite repetition of $(\dot{s}, \dot{a})$. Using these representatives, I construct a finite-horizon rewards objective $\tilde{\kappa}_{\epsilon'}$ of horizon H. The construction assigns each infinite-length sequence with the value of the original κ at the chosen representative. That is, let $w[:H]$ denote the length-H prefix of w, I define $\tilde{\kappa}_{\epsilon'}$ as:

$$\tilde{\kappa}_{\epsilon'}(w) \triangleq \kappa([w[:H]; (\dot{s}, \dot{a})^\omega]), \ \forall w \in (S \times A)^\omega.$$

By construction, $\tilde{\kappa}_{\epsilon'}$ is ϵ'-close to κ, meaning that for any infinite-length input, their evaluations differ by at most ϵ':

$$|\tilde{\kappa}_{\epsilon'}(w) - \kappa(w)| \leq \epsilon', \ \forall w \in (S \times A)^\omega.$$

Thus, an ϵ'-optimal policy for $\tilde{\kappa}_{\epsilon'}$ is $2\epsilon'$-optimal for κ.

I then reduce the approximated objective $\tilde{\kappa}_\epsilon$, which assigns a history-dependent reward at the horizon H, into a finite-horizon cumulative rewards objective, which assigns a history-independent reward at each step. To that end, I lift the state space to $U = \bigcup_{t=1}^{H} (S \times A)^t$. Each state $u_t = (S \times A)^t$at step t in the lifted state space is the length-t history of states and actions encountered in the environment. For any state before step H, I assign a reward of zero. For any state $u_H = (S \times A)^H$ at step H, I assign a reward of $\tilde{\kappa}_\epsilon([u_H; (\dot{s}, \dot{a})^\omega])$. The lifted state space and the history-independent reward function above form the desired finite-horizon cumulative rewards problem.

Dann et al. [38] introduced ORLC, a PAC reinforcement-learning algorithm for finite-horizon cumulative rewards problems.[2] Applying ORLC to the above finite-horizon cumulative rewards problem produces an $2\epsilon'$-optimal policy for κ. Finally, for any ϵ, choosing $\epsilon' = \frac{\epsilon}{2}$ gives a κ-PAC reinforcement-learning algorithm for κ. □

Environment-generic Objectives. Theorem 5.3.3 states a sufficient condition for when an environment-specific objective is PAC-learnable. The following corollary generalizes the

[2]ORLC provides an individual policy certificates (IPOC) guarantee. Dann et al. showed that IPOC implies my PAC definition, which they called "supervised-learning style PAC".

condition to environment-generic objectives.

Corollary 5.3.4. *An environment-generic objective ξ is ξ-PAC-learnable in the information-theoretic setting if ξ is uniformly continuous.*

To prove Corollary 5.3.4, I first observe the following lemma, which I prove in Section 5.7.

Lemma 5.3.5. *If an environment-generic objective is uniformly continuous, then, for all labeling functions, the induced environment-specific objective is also uniformly continuous.*

With Lemma 5.3.5, Corollary 5.3.4 is straightforward. Since each induced environment-specific objective κ is uniformly continuous, each κ is κ-PAC-learnable by Theorem 5.3.3. Thus, the objective ξ is ξ-PAC-learnable by definition.

5.3.3 Computability

I now define the computability of an objective $f\colon X^\omega \to \mathbb{R}$. The standard definition of computability of such functions depends on Type-2 Turing machines [51, Chapter 2, Definition 2] and a representation of the reals by an infinite sequence of rational approximations, called the Cauchy-representation [51, Chapter 3, Definition 3]. Recall that a Type-2 Turing machine is a Turing machine with an infinite-length input tape and a one-way infinite-length output tape. It reads the input tape and computes forever writing to the output tape.

Definition 5.3.6 (Computable Objective)**.** *An objective f is computable if a Type-2 Turing machine reads w from the input tape and writes a fast-converging Cauchy sequence $[q_0, q_1, \dots] \in \mathbb{Q}^\omega$ of rational approximations to $f(w)$ to the output, that is: $\forall n \in \mathbb{N}$, $|f(w) - q_n| \leq 2^{-n}$.*

When proving computability, this definition is tedious to work with since it requires implementing the function on a Turing machine. Instead, I will use pseudocode to formulate an algorithm that takes in an infinite-stream input w and a natural number n and outputs the n-th rational approximation q_n. Repeatedly invoking the algorithm by enumerating n produces the Cauchy sequence of rational approximations.

A classic result in computable analysis is that computable functions are continuous [51, Theorem 2.5 and 4.3]. Since an objective's domain is compact, by the Heine-Cantor theorem, this result also holds for uniform continuity. Even stronger, the following theorem, modified from Weihrauch [51, Theorem 6.2.7] for my context, guarantees that for a computable objective, for any rational $\epsilon > 0$, I can compute a horizon H that satisfies the definition of uniform continuity. Define *the modulus of continuity* of an objective as a function $m\colon \mathbb{Q} \to \mathbb{N}$

that satisfies $\forall \epsilon \in \mathbb{Q}, \forall w_1 \in X^\omega, \forall w_2 \in X^\omega : L_{\text{prefix}}(w_1, w_2) \geq m(\epsilon) \Rightarrow |f(w_1) - f(w_2)| \leq \epsilon$.

Theorem 5.3.7. *A computable objective is uniformly continuous. Further, its modulus of continuity m is computable.*

For completeness, Section 5.8 gives pseudocode that computes the modulus of continuity for any computable objective specified by the interface described above.

5.3.4 Computability Implies PAC-learnability

I now extend my result in Section 5.3.2 from the information-theoretic to the computation-theoretic setting. The following theorem states this extension.

Theorem 5.3.8. *An (environment-generic or environment-specific) objective f is f-PAC-learnable in the computation-theoretic setting if f is computable.*

Proof. Combining theorems in Section 5.3.2 and Theorem 5.3.7, a computable objective f is uniformly continuous, therefore f-PAC-learnable in the information-theoretic setting. In the computational-theoretic setting, I need to further construct a computable reinforcement-learning algorithm. Note that my proof of Corollary 5.3.4 is already constructive of an algorithm. However, I need to:

- compute the bound H from the given ϵ' and
- computably evaluate the approximated objective $\tilde{\kappa}_{\epsilon'}$.

A computable objective resolves both points:

- By Theorem 5.3.7, the bound H is computable for any ϵ.
- Evaluating the approximate objective is computable, since the approximated objective only depends on the length-H prefix of the input. □

For completeness, Section 5.9 provides the pseudocode for an f-PAC reinforcement-learning for any computable objective f.

5.4 Theorem Applications

This section applies the core theorem and corollary to three classes of objectives in the existing literature and proves each objective's PAC-learnability.

5.4.1 Reward Machine

The first class of objectives I consider is reward machines [23].

Proof of PAC-learnability I prove that the reward-machine objective reviewed in Section 3.1 is PAC-learnable.

Proposition 5.4.1. *The objective $[\![R]\!]$ of a simple reward machine R is $[\![R]\!]$-PAC-learnable.*

Proof. By Theorem 5.3.8, it is sufficient to show that a simple reward-machine objective is computable. Consider the pseudocode with Python-like syntax in Listing 5.5.

Listing 5.5 defines an algorithm for computing the simple reward-machine objective. It first initializes the state variable `u` to the initial state u_0. It then computes a horizon $H = (\lfloor \log_2(1-\gamma) \rfloor - n - \lceil \log_2 r_{\max} \rceil) / \lceil \log_2 \gamma \rceil$, where $r_{\max} = \max(|\delta_r(\cdot)|)$ is the maximum magnitude of all possible rewards. It iterates through the first H indices of the input and transits the reward machine's state according to the transitions δ_u. For each input w and n, the algorithm accumulates the discounted cumulative rewards truncated to the first H-terms: $\sum_{k=0}^{H-1} \gamma^k \delta_r(u_k, u_{k+1})$.

By definition of a computable objective, I need to show that the yield values for all n form a fast-converging Cauchy sequence:

$$\forall n \in \mathbb{N}, |\texttt{simple\_reward\_machine}(w)[n] - [\![R]\!](w)| \leq 2^{-n},$$

where I used the notation $\texttt{simple\_reward\_machine}(w)[n]$ to denote the n-th yielded output of the `simple_reward_machine` function. Then, let Δ be the difference between the algorithm's output and the objective's value:

$$\Delta \triangleq |\texttt{simple\_reward\_machine}(w)[n] - [\![R]\!](w)| = \left| \sum_{k=H}^{\infty} \gamma^k \delta_r(u_k, u_{k+1}) \right|.$$

Then, I have $\Delta \leq {r_{\max} \cdot \gamma^H}/{1-\gamma}$ by upper bounding the rewards by $r_{\max}$, then simplifying the infinite sum into a closed form. By plugging in the value of H and simplifying the inequality, I have $\Delta \leq 2^{-n}$. Thus, the objective is computable and $[\![R]\!]$-PAC-learnable. □

5.4.2 LTL Surrogate Objectives

Linear temporal logic (LTL) objectives are measurable Boolean objectives that live in the first two-and-half levels of the Borel hierarchy [43]. Various works [1, 16, 17] considered LTL objectives for reinforcement learning and empirical algorithms for learning. A common

```
# Given reward machine (U, δ_u, δ_r, u_0, γ)
def simple_reward_machine(w: Iterator[2^Π]) -> Real:
  u: U, value: Rational = u_0, 0
  rmax: Rational = max(abs(δ_r(u1, u2)) for u1, u2 in U^2)
  for n in count(0):
    H: Natural = (log2floor(1 - γ) - n - log2ceil(rmax)) / log2ceil(γ)
    for k in range(H):
        u_ = δ_u(u, w[k])
        value += γ**k * δ_r(u, u_)
        u = u_
    yield value
```

Listing 5.5: Computation of the simple reward objective

pattern of these algorithms is that they convert a given LTL formula to an intermediate specification that takes in additional hyper-parameters. They show that in an unreachable limit of these hyper-parameters, the optimal policy for this intermediate specification becomes the optimal policy for the given LTL formula. I call such intermediate specifications *LTL surrogate specifications*. Due to space, I will focus on Bozkurt et al. [1] and give the objective specified by their LTL surrogate -specification. I will show that this objective is PAC-learnable. The same process, namely writing down the LTL surrogate specification and then proving that the specified objective is PAC-learnable, also applies to the approaches in Sadigh et al. [2], Hahn et al. [16], and Hasanbeig et al. [17].

Bozkurt et al.'s LTL Surrogate Specification Given an LTL formula, Bozkurt et al. first convert the formula into a *limit deterministic Büchi Automaton (LDBA)* by a standard conversion algorithm [52] with two additional discount factor parameters. An LDBA is a non-deterministic finite automaton. It is bipartite by two sets of states, those in an *initial component* and those in an *accepting component*. Transitions in the automaton can only go from the initial component to the accepting component, but not the reverse. An LDBA is "deterministic in the limit": it only has non-deterministic ϵ-transitions in the initial component, but it is deterministic in the accepting component. The formal definition of LDBA is:

Definition 5.4.2 (LDBA)**.** *For an LTL formula over propositions* Π*, an LDBA converted from the formula is a tuple* $(U, \mathcal{E}, \delta_u, u_0, B)$*, where*

- U *is a finite set of states,*
- $\mathcal{E}$ *is a set of* ϵ*-transitions,*

- $\delta_u \colon \Big(U \times (2^{\Pi} \cup \{\epsilon\})\Big) \to 2^U$ *is a non-deterministic transition function,*
- u_0 *is an initial state, and*
- $B \subseteq U$ *is a set of accepting states.*

Additionally, U *has a bi-partition of an initial component with states* U_I *and an accepting component with states* U_B*. An LDBA satisfies the conditions: (1)* $\delta_u(u, \epsilon) = \emptyset$ *for all* $u \in U_B$*, (2)* $\delta_u(u, 2^{\Pi}) \subseteq U_B$ *for all* $u \in U_B$*, and (3)* $B \subseteq U_B$*.*

The agent and environment models are similar to a simple reward machine. In particular, at each step, the agent chooses an environment's action and steps in the environment. A labeling function classifies the current state of the environment to a tuple of truth values of the set of propositions Π.

At each step, an LDBA takes either a non-deterministic ϵ-transitions (if such transition is available) or the transition along the tuple of the truth values of the propositions. Each time the LDBA enters an accepting state, the agent receives a reward of $1 - \gamma_1$, and discounts all future rewards by γ_1. Each time the LDBA enters a non-accepting state, the agent receives no reward and discounts all future rewards by γ_2. An oracle controls the ϵ-transitions. In words, the objective is to maximize the (state-dependent) discounted cumulative rewards, assuming the oracle makes the optimal choice that maximizes the cumulative rewards.

Bozkurt et al.'s LTL Surrogate Objective Bozkurt et al.'s LTL surrogate specification is a tuple (L, γ_1, γ_2): the LDBA L and the two hyper-parameters $\gamma_1, \gamma_2 \in \mathbb{Q}$. It specifies an environment-generic objective $[\![(L, \gamma_1, \gamma_2)]\!] \colon (2^{\Pi})^{\omega} \to \mathbb{R}$. Let $\mathcal{E}^+ = \mathcal{E} \cup \{\bot\}$, where $\mathcal{E}$ is the set of ϵ-transitions and $\bot$ is a non-ϵ-transition (i.e., following a transition with a tuple

classified by the labeling function), the objective is:

$$
\begin{aligned}
&[\![(L,\gamma_1,\gamma_2)]\!](w) = \max_{w_\mathcal{E}\in\mathcal{E}^\omega} g(w_\mathcal{E},w) \quad \text{where}\\
&g(w_\mathcal{E},w) = \sum_{i=1}^{\infty} R(u_i)\prod_{j=1}^{i-1}\Gamma(u_j),\\
&R(u) = (1-\gamma_1)\mathbb{1}\{u\in B\},\\
&\Gamma(u) = \gamma_1\mathbb{1}\{u\in B\} + \gamma_2\mathbb{1}\{u\notin B\},\\
&\forall k\geq 0\colon u_{k+1} = \delta_u(u_k, w_k^+),\\
&\quad t_k = \sum_{i=1}^{k}\mathbb{1}\{w_\mathcal{E}[i]=\bot \text{ or } (u_k,w_\mathcal{E}[i])\notin\delta_u\},\\
&\quad w_k^+ = \begin{cases} w[t_k] & \text{if } w_\mathcal{E}[k]=\bot \text{ or } (u_k,w_\mathcal{E}[k])\notin\delta_u\\ w_\mathcal{E}[k] & \text{otherwise}\end{cases}
\end{aligned}
\qquad . \tag{5.2}
$$

Here, t_k is the step count of the environment when the LDBA takes its k-th step. Note that $t_k \leq k$, since the environment does not step when LDBA takes an ϵ-transition. The value $w[t_k]$ is the tuple of truth values of the input infinite-length sequence w at t_k. I define w_k^+ as the transition label taken by the LDBA at the k-th step: It is either (1) a tuple of truth values $w[k]$, if $w_\mathcal{E}[k]$ is a non-ϵ-transition or an ϵ-transition that is not available from the current LDBA state u_k, or (2) the ϵ-transition $w_\mathcal{E}[k]$. By its definition, w_k^+ is always a valid transition of the LDBA, and it always leads to a deterministic next state. Therefore, I write $u_{k+1} = \delta_u(u_k, w_k^+)$ to denote that the LDBA state u_{k+1} follows this deterministic transition to transit to the next state.

Proof of PAC-learnability I now prove that the objective specified by an LTL surrogate specification in Bozkurt et al. [1] is PAC-learnable. Although this section aims to show an example, as I mentioned, the proof strategy here also applies to the approaches in Sadigh et al. [2], Hahn et al. [16], and Hasanbeig et al. [17].

Proposition 5.4.3. *Bozkurt et al.'s LTL surrogate objective* $[\![(L,\gamma_1,\gamma_2)]\!]$ *is* $[\![(L,\gamma_1,\gamma_2)]\!]$*-PAC-learnable.*

Proof. By Theorem 5.3.8, it is sufficient to show that Bozkurt et al.'s objective is computable. Consider the pseudocode with Python-like syntax in Listing 5.6.

Listing 5.6 gives pseudocode for computing Bozkurt et al.'s objective. The pseudocode contains two procedures. The procedure `bozkurt_helper` computes g but truncates the sum to the first $H = \left(\lfloor \log_2(1-\max(\gamma_1,\gamma_2)\rfloor - n\right) / \lceil \log_2 \max(\gamma_1,\gamma_2)\rceil$ terms. The proce-

```
# Given LDBA (U, ℰ, δ_u, u_0, B) and γ_1, γ_2
def bozkurt_objective(w: Iterator[2^ℕ]) -> Real:
  for n in count(0):
    gamma_max: Rational = max(γ_1, γ_2)
    H: Natural = (log2floor(1 - gamma_max) - n) / log2ceil(gamma_max)
    v: Rational = 0
    for w_e in ℰ^H:
      v = max(v, bozkurt_helper(H, w_e, w))
    yield v

def bozkurt_helper(H: Natural, w_e: ℰ^H, w: S^ω) -> Rational:
  v: Rational, u: U, discount: Rational = 0, u_0, 1

  for k in range(H):
    if u in B:
        reward, gamma = 1, γ_1
    else:
        reward, gamma = 0, γ_2

    v += reward * discount
    discount *= gamma

    if w_e[k] == ⊥ or (u, w_e[k]) not in δ_u:
        w_k_plus = w[k]
    else:
        w_k_plus = w_e[k]

    u = δ_u(u, w_k_plus)

  return v
```

Listing 5.6: Computation of Bozkurt et al.'s objective

dure `bozkurt_objective` then computes the n-th rational approximation of the objective's value. It invokes the helper function for all $\hat{w}_\epsilon \in \mathcal{E}^n$ and calculates the value of $\max_{\hat{w}_\epsilon \in \mathcal{E}^n} \texttt{bozkurt\_helper}(\hat{w}_\epsilon, w, n)$.

Section 5.11 proves that the yielded values of `bozkurt_objective` for all $n \in \mathbb{N}$ form a fast-converging Cauchy sequence:

$$|\texttt{bozkurt\_objective}(w)[n] - [\![(L, \gamma_1, \gamma_2)]\!](w)| \leq 2^{-n}.$$

Therefore, the objective is computable and consequently $[\![(L, \gamma_1, \gamma_2)]\!]$-PAC-learnable. □

5.4.3 Geometric Linear Temporal Logic

Littman et al. [21] introduced geometric linear temporal logic (GLTL), a variant of linear temporal logic with expiring temporal operators. This section formalizes the objective specified by a GLTL formula and proves that the objective is PAC-learnable.

5.4.3.1 GLTL Specification

A GLTL formula is built from a finite set of atomic propositions Π, logical connectives $\neg, \wedge, \vee$, temporal next X, and expiring temporal operators G_θ (expiring always), F_θ (expiring eventually), and U_θ (expiring until). Equation (5.3) gives the grammar of a GLTL formula ϕ over the set of atomic propositions Π:

$$\phi := a \;\Big|\; \neg\phi \;\Big|\; \phi \wedge \phi \;\Big|\; \phi \vee \phi \;\Big|\; \mathsf{X}\phi \;\Big|\; \mathsf{G}_\theta\phi \;\Big|\; \mathsf{F}_\theta\phi \;\Big|\; \phi\, \mathsf{U}_\theta\, \phi,\ a \in \Pi, \theta \in \mathbb{Q}. \tag{5.3}$$

Each temporal operator (i.e., G, F and U) has a rational expiration probability θ in range $(0, 1)$. For example, $\mathsf{F}_{0.9} \mathit{goal} \wedge \mathsf{G}_{0.9} \mathit{lava}$ is a valid GLTL formula.

The semantics of GLTL is similar to that of LTL (which I review in Section 4.2.1), except that each operator expires at every step with the given probability θ associated with the operator. In particular, for the expiring operator $\mathsf{G}_\theta\phi$, if ϕ is always true prior to an expiration event, then the overall formula evaluates to true; otherwise, ϕ is ever false prior to the expiration event, then the overall formula evaluates to false. Conversely, for the expiring operator $\mathsf{F}_\theta\phi$, if ϕ is ever true prior to an expiration event, then the overall formula evaluates to true; otherwise, ϕ is always false prior to the expiration event, then the overall formula evaluates to false. I give the formal semantics of GLTL below.

I first define the *event form* of a GLTL formula. An event-form of a GLTL formula is an LTL formula. This LTL formula contains the propositions in the GLTL formula and an additional set of propositions called *expiration events*. I define the event-form of a GLTL formula in such a way that, when an expiration event triggers at time t, the entire subformula corresponding to this event expires. The event form $\mathcal{T}(\phi)$ of a GLTL formula ϕ is

defined recursively as:

$$\mathcal{T}(\phi) \triangleq \begin{cases} \neg\mathcal{T}(\psi) & \phi = \neg\psi \\ \mathcal{T}(\psi_1) \wedge \mathcal{T}(\psi_2) & \phi = \psi_1 \wedge \psi_2 \\ \mathcal{T}(\psi_1) \vee \mathcal{T}(\psi_2) & \phi = \psi_1 \vee \psi_2 \\ \mathsf{X}\mathcal{T}(\psi) & \phi = \mathsf{X}\psi \\ \mathcal{T}(\psi)\ \mathsf{U}\ (\mathcal{T}(\psi) \wedge e_\phi) & \phi = \mathsf{G}_\theta\psi \\ \neg e_\phi\ \mathsf{U}\ \mathcal{T}(\psi) & \phi = \mathsf{F}_\theta\psi \\ (\mathcal{T}(\psi_1) \wedge \neg e_\phi)\ \mathsf{U}\ \mathcal{T}(\psi_2) & \phi = \psi_1\ \mathsf{U}_\theta\ \psi_2 \end{cases}$$

Here, each e_ϕ is a fresh proposition corresponding to the sub-formula ϕ. In words:

- If the outer formula is logical connective or temporal next, the operator $\mathcal{T}$ recursively converts the sub-formula(s) while preserving the outer formula's operator.
- If the outer formula is G_θ, it recursively converts the sub-formula. It outputs the LTL formula $\mathcal{T}(\psi)\ \mathsf{U}\ (\mathcal{T}(\psi_2) \wedge e_\phi)$ that requires the converted sub-formula to hold until the expiration event e_ϕ expires.
- If the outer formula is F_θ, it recursively converts the sub-formula. It outputs the LTL formula $\neg\mathcal{T}(\psi)\ \mathsf{U}\ e_\phi$ that requires the converted sub-formula to become true at least once before the expiration event expires.
- If the outer formula is $\psi_1\,\mathsf{U}_\theta\,\psi_2$, it recursively converts the sub-formula and outputs the LTL formula $(\mathcal{T}(\psi_1) \wedge \neg e_\phi)\ \mathsf{U}\ \mathcal{T}(\psi_2)$ that requires the converted sub-formula $\mathcal{T}(\psi_1)$ to hold until $\mathcal{T}(\psi_2)$ becomes true, all before the expiration event expires.

For example, the event form of the GLTL formula $\phi = \mathsf{G}_{0.9}(a \wedge \mathsf{F}_{0.9}b)$ is

$$\mathcal{T}(\phi) = (a \wedge (\neg e_\psi\ \mathsf{U}\ b))\ \mathsf{U}\ ((a \wedge (\neg e_\psi\ \mathsf{U}\ b)) \wedge e_\phi)\,, \text{ where } \psi \text{ is the sub-formula } \mathsf{F}_{0.9}b.$$

During an evaluation of a GLTL formula, each expiration event e_ϕ corresponds to an infinite stream of independently distributed Bernoulli random variables, each triggering with probability θ, the expiration probability associated with the outermost expiring temporal operator of ϕ. Given an infinite-length path of propositions w, the probability of w satisfying a GLTL formula ϕ is the probability of w satisfying the event-form LTL formula $\mathcal{T}(\phi)$, with stochasticity due to the expiration events.

5.4.3.2 GLTL Objective

I now give the objective specified by a GLTL formula. A GLTL formula ϕ over propositions Π specifies an environment-generic objective $[\![\phi]\!]\colon (2^{\Pi})^{\omega} \to \mathbb{R}$, given by:

$$[\![\phi]\!](w) = \mathsf{P}_{\boldsymbol{e}\sim\text{Bernoulli}(\boldsymbol{\theta})}((zip\,(w, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi))). \tag{5.4}$$

Here, $\boldsymbol{e}$ is the set of all the expiration events in ϕ, and each is a random variable following an infinite stream of Bernoulli distribution. I use $zip\,(w, w_{\boldsymbol{e}}) : (2^{\Pi\cup\boldsymbol{e}})^{\omega}$ to denote the element-wise combination of w and the $w_{\boldsymbol{e}}$. I write $zip\,(w, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi)$ to denote that the infinite-length path $zip\,(w, w_{\boldsymbol{e}})$ satisfies the LTL formula $\mathcal{T}(\phi)$ according to the LTL semantics, which I review in Section 4.2.1. The objective is then the probability of this infinite-length path satisfying the LTL formula $\mathcal{T}(\phi)$, with stochasticity due to the expiration events.

5.4.3.3 Proof of PAC-learnability

I now prove that the objective specified by a GLTL formula is PAC-learnable.

Proposition 5.4.4. *The objective $[\![\phi]\!]$ specified by a GLTL formula ϕ is $[\![\phi]\!]$-PAC-learnable.*

Proof Outline The strategy of my proof is as follows.

- First, I will prove that a GLTL objective is uniformly continuous. In particular, for any $\epsilon \in \mathbb{Q}$, I will give an upper bound H, so that the objective maps infinite-length paths inputs sharing a prefix of H to ϵ-close values.
- Next, I present a pseudocode that computes a GLTL objective. To obtain the n-th rational approximation, the pseudocode first determines the upper bound H from $\epsilon = 2^{-n}$. It then computes a lower bound on the objective value after observing the first H indices of the input and returns this lower bound as the n-th rational approximation.
- Finally, I show that the rational approximations form a fast-converging Cauchy sequence, which proves that the objective is computable. Therefore by Theorem 5.3.7, the objective is PAC-learnable.

GLTL Objective is Uniformly Continuous First, the following lemma states that if all expiration events trigger simultaneously at step H, then the evaluation of the event-form of the GLTL formula depends only on the length-H prefix of the infinite-length input.

Lemma 5.4.5 (Simultaneous Expiration)**.** *Given a GLTL formula ϕ, the satisfaction of its event-form depends on the input infinite-length path up to a horizon that all expiration events are simultaneously triggered. That is:*

$$\forall w_{\boldsymbol{e}} \in (2^{e})^{\omega}.\forall w_1 \in (2^{\Pi})^{\omega}.\forall w_2 \in (2^{\Pi})^{\omega}:$$
$$\textit{if } \left(w_{\boldsymbol{e}}[H] = \vec{1} \wedge w_1[{:}H] = w_2[{:}H]\right) \textit{ then } \left(zip\,(w_1, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi) \Leftrightarrow zip\,(w_2, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi)\right)$$

Then, I utilize Lemma 5.4.5 to prove the following lemma that a GLTL objective is uniformly continuous.

Lemma 5.4.6. *A GLTL objective $[\![\phi]\!]$ is uniformly continuous.*

Proof. For any input word w, I rewrite the value of the objective in Equation (5.4) by unrolling the first H steps. Then, I condition the objective (which is a probability) on if all expirations expire simultaneously at each step. In particular, let E_k denote the event that all expirations trigger simultaneously at step k and let $\neg E_{1\ldots H}$ denote the event that all expiration never simultaneously trigger before step H. I expand Equation (5.4) as:

$$\begin{aligned}[\![\phi]\!](w) = \sum_{k=1}^{H} \mathsf{P}(E_k) \cdot \mathsf{P}(zip\,(w, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi) \mid E_k) + \\ \mathsf{P}(\neg E_{1\ldots H}) \cdot \mathsf{P}(\, zip\,(w, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi) \mid \neg E_{1\ldots H}\,)\end{aligned} \tag{5.5}$$

Then, consider two paths w_1 and w_2 sharing a prefix of length H. By Lemma 5.4.5, the satisfaction of $\mathcal{T}(\phi)$ depends only on the first H steps of w_1 and w_2. Thus, for all $0 \leq k \leq H$, $\mathsf{P}(zip\,(w_1, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi) \mid E_k) = \mathsf{P}(zip\,(w_2, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi) \mid E_k)$. Therefore, for the difference $\Delta = |[\![\phi]\!](w_1) - [\![\phi]\!](w_2)|$, I cancel the first H terms in the sum to get:

$$\begin{aligned}\Delta &= \mathsf{P}(\neg E_{1\ldots H}) \cdot |\mathsf{P}(\, zip\,(w_1, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi) \mid \neg E_{1\ldots H}\,) - \mathsf{P}(\, zip\,(w_2, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi) \mid \neg E_{1\ldots H}\,)| \\ &\leq \mathsf{P}(\neg E_{1\ldots H})\end{aligned}$$

Since $\mathsf{P}(\neg E_{1\ldots H})$ is the probability of the all the expirations not triggering simultaneously for the first H steps, I have: $\mathsf{P}(\neg E_{1\ldots H}) = (1 - \prod_{\theta_i \in \phi} \theta_i)^H$. Consequently, I obtain the upper bound $\Delta \leq (1 - \prod_{\theta_i \in \phi} \theta_i)^H$.

Given any $\epsilon > 0$, I can always choose $H = \frac{\log(\epsilon)}{\log(1 - \prod_{\theta_i \in \phi} \theta_i)}$ so that $\Delta \leq \epsilon$. Therefore, $[\![\phi]\!]$ is uniformly continuous. □

Computation of a GLTL Objective I give pseudocode for computing a GLTL objective in Listing 5.7. The code in Listing 5.7 first computes a horizon $H = \frac{-n}{\lceil \log_2(1 - \prod_{\theta_i \in \phi} \theta_i) \rceil}$. It then

```
# Given GLTL formula ϕ
def GLTL(w: Iterator[2^Π]) -> Real:
  for n in count(0):
    theta = prod(ϕ.thetas) # take product of all the expiration probabilities in |ϕ|
    H = -n / log2ceil(1 - theta)
    v = 0
    for w_e in e^H:
      if w_e has a simultaneous trigger:
        wp = [zip(w[: H], w_e); ·^ω]
        if wp ⊨ T(ϕ):
          e_prob = 1
          for e_i in e: # enumerates all the events in the formula
            w_e_i = w_e[e_i]
            for j in range(H): # enumerates the values in the length-H stream of an event
              e_prob *= w_e_i.theta if w_e_i[j] == 1 else (1 - w_e_i.theta) # event
              ↪ triggers with probability w_e_i's theta
          v += e_prob
    yield v
```

Listing 5.7: Computation of a GLTL objective

computes the sum of H-terms in Equation (5.5):

$$\sum_{k=1}^{H} \mathsf{P}(E_k) \cdot \mathsf{P}(zip\,(w, w_{\boldsymbol{e}}) \vDash \mathcal{T}(\phi) \mid E_k). \tag{5.6}$$

To compute this sum, it enumerates all combinations of expiration events of length H. If a simultaneous expiration ever happens in this length H events, it tests if the input word is accepted or rejected by $\mathcal{T}(\phi)$. In particular, to perform the test, it forms any eventual cyclic path $w' = [zip\,(w[: H], w_{\boldsymbol{e}})\,; \cdot^{\omega}]$ that starts with the first H indices of the input w and the length-H sequence of events $w_{\boldsymbol{e}}$ and ends in an arbitrarily chosen cycle. It then test if w' satisfies $\mathcal{T}(\phi)$ by a standard LTL model checking algorithm [42, Chapter 5.2]. Due to Lemma 5.4.5, this test is equivalent to testing if $zip\,(w, w_{\boldsymbol{e}})$ satisfies $\mathcal{T}(\phi)$. Finally, it sums the probability of all length-H sequence of expiration events that pass this test. This produces the desired sum in Equation (5.6).

Conclusion By the proof of Lemma 5.4.6, the sum in Equation (5.6) is at most $\epsilon = 2^{-n}$ smaller than the objective's value. Therefore, the return values of Listing 5.7 for all $n \in \mathbb{N}$ form a fast-converging Cauchy sequence that converges to the objective's value. Thus, the objective is computable and $[\![\phi]\!]$-PAC-learnable.

5.5 Proof of Lemma 5.4.5

I perform the proof inductively. Specifically, given any GLTL formula ϕ, I pose the inductive hypothesis that each sub-formula of ϕ satisfies Lemma 5.4.5. Given this inductive hypothesis, I prove that ϕ also satisfies Lemma 5.4.5.

If the outer formula of ϕ is $\neg\psi$ Suppose that w_e is true at step H and that w_1 and w_2 match up to step H:

$$w_e[H] = \vec{1} \wedge w_1[{:}H] = w_2[{:}H].$$

By the induction hypothesis, the evaluations of the sub-formula ψ are the same for the two inputs w_1 and w_2:

$$zip\,(w_1, w_e) \vDash \mathcal{T}(\psi) \Leftrightarrow zip\,(w_2, w_e) \vDash \mathcal{T}(\psi).$$

Therefore, I can prepend negations to both sides:

$$zip\,(w_1, w_e) \vDash \neg\mathcal{T}(\psi) \Leftrightarrow zip\,(w_2, w_e) \vDash \neg\mathcal{T}(\psi).$$

By definition of $\mathcal{T}$, I may drop the negations: $zip\,(w_1, w_e) \vDash \mathcal{T}(\phi) \Leftrightarrow zip\,(w_2, w_e) \vDash \mathcal{T}(\phi)$, which proves this case.

If the outer formula of ϕ is $\psi_1 \wedge \psi_2$ or $\psi_1 \vee \psi_2$ I will prove for the case of $\phi = \psi_1 \wedge \psi_2$. The case of $\phi = \psi_1 \vee \psi_2$ is the same by changing the logical connective from $\wedge$ to $\vee$.

Suppose that w_e is true at step H and that w_1 and w_2 match up to step H:

$$w_e[H] = \vec{1} \wedge w_1[{:}H] = w_2[{:}H].$$

By the induction hypothesis, the evaluations of each of the sub-formula ψ_1 and ψ_2 are the same for the two inputs w_1 and w_2, that is:

$$zip\,(w_1, w_e) \vDash \mathcal{T}(\psi_1) \Leftrightarrow zip\,(w_2, w_e) \vDash \mathcal{T}(\psi_1)$$

and

$$zip\,(w_1, w_e) \vDash \mathcal{T}(\psi_2) \Leftrightarrow zip\,(w_2, w_e) \vDash \mathcal{T}(\psi_2).$$

Therefore, I can join the two equivalences by the logical connective: $zip\,(w_1, w_e) \vDash \mathcal{T}(\psi_1) \wedge \mathcal{T}(\psi_2) \Leftrightarrow zip\,(w_2, w_e) \vDash \mathcal{T}(\psi_1) \wedge \mathcal{T}(\psi_2)$. By definition of $\mathcal{T}$, I have: $zip\,(w_1, w_e) \vDash \mathcal{T}(\phi) \Leftrightarrow zip\,(w_2, w_e) \vDash \mathcal{T}(\phi)$, which proves this case.

If the outer formula of ϕ is $\mathsf{G}_\theta\psi$ Suppose that $w_{\boldsymbol{e}}$ is true at step H and that w_1 and w_2 match up to step H:

$$w_{\boldsymbol{e}}[H] = \vec{1} \wedge w_1[:H] = w_2[:H].$$

Consider the infinite-length paths $w_1^i \triangleq w_1[i:]$, $w_2^i \triangleq w_2[i:]$ and $w_{\boldsymbol{e}}^i \triangleq w_{\boldsymbol{e}}[i:]$, the suffixes of w_1, w_2 and $w_{\boldsymbol{e}}$ beginning at some step i where $0 \leq i \leq H$. The expiration event triggers at step $H - i$ for $w_{\boldsymbol{e}}[i:]$, that is: $w_{\boldsymbol{e}}^i[H-i] = \vec{1}$. Further, w_1^i and w_2^i match up to length $H - i$, that is: $w_1^i[H-i:] = w_2^i[H-i:]$. Therefore, by the induction hypothesis, the evaluations of the sub-formula ψ are the same for all pairs of suffix inputs w_1^i and w_2^i for all i, that is:

$$\forall 0 \leq i \leq H. zip\left(w_1, w_{\boldsymbol{e}}\right)[i:] \vDash \mathcal{T}(\psi) \Leftrightarrow zip\left(w_2, w_{\boldsymbol{e}}\right)[i:] \vDash \mathcal{T}(\psi). \tag{5.7}$$

Since $w_{\boldsymbol{e}}$ is fixed on both sides, conjuncting both sides with e_ϕ, I also have:

$$\forall 0 \leq i \leq H. zip\left(w_1, w_{\boldsymbol{e}}\right)[i:] \vDash \left(\mathcal{T}(\psi) \wedge e_\phi\right) \Leftrightarrow zip\left(w_2, w_{\boldsymbol{e}}\right)[i:] \vDash \left(\mathcal{T}(\psi) \wedge e_\phi\right). \tag{5.8}$$

To summarize Equations (5.7) and (5.8): The satisfaction relations in the above equations are equal between w_1 and w_2 up to step H.

By defintion of $\mathcal{T}$ and the semantics of LTL (reviewed in Section 4.2.1), $zip\left(w, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ if and only if:

$$\exists j \geq 0, zip\left(w, w_{\boldsymbol{e}}\right)[j:] \vDash \left(\mathcal{T}(\psi) \wedge e_\phi\right) \text{ and } \forall k.\, 0 \leq k < j \Rightarrow zip\left(w, w_{\boldsymbol{e}}\right)[k:] \vDash \mathcal{T}(\psi). \tag{5.9}$$

If $zip\left(w_1, w_{\boldsymbol{e}}\right)[j:] \vDash \left(\mathcal{T}(\psi) \wedge e_\phi\right)$ and $zip\left(w_1, w_{\boldsymbol{e}}\right)[j:] \vDash \left(\mathcal{T}(\psi) \wedge e_\phi\right)$ are both true for any $j \leq H$, then $zip\left(w_1, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ equals $zip\left(w_2, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ due to all satisfaction relations in Equation (5.9) match between w_1 and w_2 up to step H.

On the other hand, if $zip\left(w_1, w_{\boldsymbol{e}}\right)[j:] \vDash \left(\mathcal{T}(\psi) \wedge e_\phi\right)$ and $zip\left(w_1, w_{\boldsymbol{e}}\right)[j:] \vDash \left(\mathcal{T}(\psi) \wedge e_\phi\right)$ are never true for some $j \leq H$, $zip\left(w_1, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ and$zip\left(w_2, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ both equal false. That is because, since e_ϕ is true at step H, in order for $zip\left(w, w_{\boldsymbol{e}}\right)[H:] \vDash \left(\mathcal{T}(\psi) \wedge e_\phi\right)$ to be false, $\mathcal{T}(\psi)$ must be false at step H — which in turn implies that there is no j such that $\mathcal{T}(\psi)$ will hold until $\mathcal{T}(\psi) \wedge e_\phi$ becomes true at step j.

In both scenarios, I have $zip\left(w_1, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ equivalent to $zip\left(w_2, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$, which proves this case.

If the outer formula of ϕ is $\psi_1 \mathsf{U}_\theta \psi_2$ Suppose that $w_{\boldsymbol{e}}$ is true at step H and that w_1 and w_2 match up to step H:

$$w_{\boldsymbol{e}}[H] = \vec{1} \wedge w_1[:H] = w_2[:H].$$

Consider the infinite-length paths $w_1^i = w_1[i{:}]$, $w_2^i = w_2[i{:}]$ and $w_{\boldsymbol{e}}^i = w_{\boldsymbol{e}}[i{:}]$, the suffixes of w_1, w_2 and $w_{\boldsymbol{e}}$ beginning at some step i where $0 \leq i \leq H$. The expiration event triggers at step $H - i$ for $w_{\boldsymbol{e}}[i{:}]$, that is: $w_{\boldsymbol{e}}^i[H-i] = \vec{1}$. Further, w_1^i and w_2^i match up to length $H - i$, that is: $w_1^i[H-i{:}] = w_2^i[H-i{:}]$. Therefore, by the induction hypothesis, the evaluations of each of the sub-formulas ψ_1 and ψ_2 are the same for all pairs of suffix inputs w_1^i and w_2^i for all i, that is:

$$\begin{aligned} \forall 0 \leq i \leq H. zip\left(w_1, w_{\boldsymbol{e}}\right)[i{:}] \vDash \mathcal{T}(\psi_1) \Leftrightarrow zip\left(w_2, w_{\boldsymbol{e}}\right)[i{:}] \vDash \mathcal{T}(\psi_1) \text{ and} \\ \forall 0 \leq i \leq H. zip\left(w_1, w_{\boldsymbol{e}}\right)[i{:}] \vDash \mathcal{T}(\psi_2) \Leftrightarrow zip\left(w_2, w_{\boldsymbol{e}}\right)[i{:}] \vDash \mathcal{T}(\psi_2). \end{aligned} \tag{5.10}$$

Since $w_{\boldsymbol{e}}$ is fixed on both sides, conjuncting both sides with e_ϕ, I also have:

$$\begin{aligned} \forall 0 \leq i \leq H. zip\left(w_1, w_{\boldsymbol{e}}\right)[i{:}] \vDash (\mathcal{T}(\psi_1) \wedge e_\phi) \Leftrightarrow zip\left(w_2, w_{\boldsymbol{e}}\right)[i{:}] \vDash (\mathcal{T}(\psi_1) \wedge e_\phi). \text{ and} \\ \forall 0 \leq i \leq H. zip\left(w_1, w_{\boldsymbol{e}}\right)[i{:}] \vDash (\mathcal{T}(\psi_2) \wedge e_\phi) \Leftrightarrow zip\left(w_2, w_{\boldsymbol{e}}\right)[i{:}] \vDash (\mathcal{T}(\psi_2) \wedge e_\phi). \end{aligned} \tag{5.11}$$

To summarize Equations (5.10) and (5.11): The satisfaction relations in the above equations are equal between w_1 and w_2 up to step H.

By definition of $\mathcal{T}$ and the semantics of LTL, $zip\left(w, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ iff:

$$\exists j \geq 0, zip\left(w, w_{\boldsymbol{e}}\right)[j{:}] \vDash \mathcal{T}(\psi_2) \text{ and } \forall k.\, 0 \leq k < j \Rightarrow zip\left(w, w_{\boldsymbol{e}}\right)[k{:}] \vDash (\mathcal{T}(\psi_1) \wedge \neg e_\phi). \tag{5.12}$$

If $zip\left(w_1, w_{\boldsymbol{e}}\right)[j{:}] \vDash \mathcal{T}(\psi_2)$ and $zip\left(w_1, w_{\boldsymbol{e}}\right)[j{:}] \vDash \mathcal{T}(\psi_2)$ are both true for any $j \leq H$, then $zip\left(w_1, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ equals $zip\left(w_2, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ due to all satisfaction relations in Equation (5.12) match between w_1 and w_2 up to step H.

On the other hand, if $zip\left(w_1, w_{\boldsymbol{e}}\right)[j{:}] \vDash \mathcal{T}(\psi_2)$ and $zip\left(w_1, w_{\boldsymbol{e}}\right)[j{:}] \vDash \mathcal{T}(\psi_2)$ are never true for some $j \leq H$, $zip\left(w_1, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ and$zip\left(w_2, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ both equal false. That is because, since e_ϕ is true at step H, $\mathcal{T}(\psi_2) \wedge \neg e_\phi$ is false at step H — which in turn implies that there is no j such that $\mathcal{T}(\psi_1) \wedge \neg e_\phi$ will hold until $\mathcal{T}(\psi_2)$ becomes true at step j.

In both scenarios, I have $zip\left(w_1, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$ equivalent to $zip\left(w_2, w_{\boldsymbol{e}}\right) \vDash \mathcal{T}(\phi)$, which proves this case.

If the outer formula of ϕ is $\mathsf{F}_\theta\psi$ By definition of the conversion operator $\mathcal{T}$, it is the case that $\mathcal{T}(\mathsf{F}_\theta\psi) = \mathcal{T}(\mathtt{true}\, \mathsf{U}_\theta\, \psi)$. Therefore the proof of the case of $\phi = \mathsf{F}_\theta\psi$ is the same as the proof of the case of $\phi = \psi_1\, \mathsf{U}_\theta\, \psi_2$, by specializing $\psi_1 = \mathtt{true}$ and $\psi_2 = \psi$.

5.6 Summary of Works on LTL Surrogate Objectives

This section briefly reviews the literature on LTL surrogate objectives. [3]

To my knowledge, Sadigh et al. [2] first proposed LTL as an objective for model-free reinforcement learning. They transformed LTL formulas into Rabin automata that give out rewards to the agent. Although the approach was appealing, Hahn et al. [16] identified counterexamples demonstrating that the translation was not entirely correct. Subsequently, both Hahn et al. [16] and Hasanbeig et al. [17] proposed to use LDBA-based reward schemes, and Hahn et al. [16]'s approach addressed the issues in Sadigh et al. [2]. Later, Bozkurt et al. [1] proposed an LDBA-based reward scheme that was less sparse than previous LDBA-based reward schemes—meaning this scheme provides rewards not only at the sink states of the LDBA, but also at intermediate states. In the same year, Hahn et al. [53] proposed a dense reward scheme and conducted experimental comparisons of various approaches.

Note that although all the above approaches attempt to use LTL as reinforcement-learning objectives, Yang, Littman, and Carbin [22] proved that PAC learning is only possible for a subset of LTL formulas called the finitary formulas. Nonetheless, approaches in the previous paragraph all fall into a common pattern that they convert a given LTL formula to an intermediate specification (Rabin automaton or LDBA) that takes in additional hyper-parameters. They show that in an unreachable limit of these hyper-parameters, the optimal policy for this intermediate specification becomes optimal for the given LTL formula. As mentioned in Section 5.4.2, I view these approaches as introducing LTL surrogate objectives and using them as proxies to the true LTL objectives.

5.7 Proof of Lemma 5.3.5

Proof. Since ξ is uniformly continuous, by definition, I have:

$$\begin{aligned} &\forall \epsilon > 0. \exists H \in \mathbb{N}. \forall w_3 \in F^\omega. \forall w_4 \in F^\omega. \\ &L_{\text{prefix}}(w_3, w_4) \geq H \Rightarrow |\xi(w_3) - \xi(w_4)| \leq \epsilon. \end{aligned}$$

Let $\kappa = \xi \circ \mathcal{L}$ be the environment-specific objective induced by ξ and the labeling function $\mathcal{L}$. By rewriting w_3 as $\mathcal{L}(w_1)$ and w_4 as $\mathcal{L}(w_2)$, I get:

$$\begin{aligned} &\forall \epsilon > 0. \exists H \in \mathbb{N}. \forall w_1 \in (S \times A)^\omega. \forall w_2 \in (S \times A)^\omega. \\ &L_{\text{prefix}}\left(\mathcal{L}\left(w_1\right), \mathcal{L}\left(w_2\right)\right) \geq H \Rightarrow |\kappa(w_1) - \kappa(w_2)| \leq \epsilon. \end{aligned} \tag{5.13}$$

[3] I acknowledge the valuable input from an anonymous reviewer that helped us with this summary.

Consider any $w \in (S \times A)^\omega$ and $w' \in (S \times A)^\omega$. If w and w' share a prefix of length H, then the labeling function also maps them to infinite-length paths sharing a prefix of length at least H, that is:

$$\forall H \in \mathbb{N}. \forall w_1 \in (S \times A)^\omega. \forall w_2 \in (S \times A)^\omega. L_{\text{prefix}}(w_1, w_2) \geq H \Rightarrow L_{\text{prefix}}(\mathcal{L}(w_1), \mathcal{L}(w_2)) \geq H. \tag{5.14}$$

By chaining the implications in Equation (5.13) and Equation (5.14), I get:

$$\forall \epsilon > 0. \exists H \in \mathbb{N}. \forall w_1 \in (S \times A)^\omega. \forall w_2 \in (S \times A)^\omega.$$
$$L_{\text{prefix}}(w_1, w_2) \geq H \Rightarrow |\kappa(w_1) - \kappa(w_2)| \leq \epsilon.$$

Therefore, κ is uniformly continuous by definition. □

5.8 Computing the Modulus-of-Continuity

Listing 5.8 gives pseudocode for computing the modulus of continuity of any computable objective given by the interface $(X^\omega \times \mathbb{N}) \to \mathbb{Q}$ (described in Section 5.3.3).

```
def modulus_of_continuity(fn: Callable[Iterator[X]][Rational], ϵ: Rational) -> Natural:
  H = 1
  while True:
    try:
      for w in X^H:
        n = log2ceil(ϵ)
        for _ in zip(range(n), fn(w)):
          pass
      return H
    except StopIteration:
      H += 1
```

Listing 5.8: Computation of the modulus-of-continuity of a computable objective

The algorithm enumerates H from 0. It forms finite-length words w for each X^H and invokes the objective on w and $n = \lceil \log_2(\epsilon) \rceil$. If the computation of the objective attempts to read $w[k]$ for some k greater than H, an exception is thrown. The exception terminates the enumeration of X^H and returns the control to the outer loop. The algorithm essentially finds the first H such that the objective only needs to inspect the first H indices of w to calculate an ϵ-close approximation to the objective's value.

5.9 PAC Reinforcement-Learning Algorithm for Computable Objectives

Listing 5.9 gives pseudocode for a reinforcement-learning algorithm for any computable objective given by the interface $(X^\omega \times \mathbb{N}) \to \mathbb{Q}$ (described in Section 5.3.3). The algorithm first computes a sufficient horizon bound H for achieving $\frac{\epsilon}{2}$-approximation to the objective's value. It then constructs the lifted MDP with finite-horizon cumulative rewards as described in the proof of Theorem 5.3.3. Finally, it invokes `rl_finite_horizon_cumulative_rewards`, an existing PAC reinforcement-learning algorithm for finite-horizon cumulative rewards problem to obtain a $\frac{\epsilon}{2}$-optimal policy. Overall, the obtained policy is ϵ-optimal for the computable objective.

5.10 Proof of Unnecessity

I complement my result and prove that my conditions are only sufficient but not necessary by giving two examples.

The first example is an objective that is not uniformly continuous (or computable) but is PAC-learnable. Consider an environment-generic objective $\xi\colon \{a, b\}^\omega \to \mathbb{R}$ with features $F = \{a, b\}$, given by:

$$\xi(w) = \mathbb{1}\{w \neq \hat{w}\} \quad \text{where } \hat{w} = abaabaaab\ldots$$

That is, the objective assigns a value of 0 for $\hat{w}$, which is an infinite-length path with a naturally increasing number of as between infinitely many bs, and 1 otherwise. Any finite state Markov chain has zero probability of generating $\hat{w}$, since the pattern of $\hat{w}$ necessarily requires an infinite memory to generate. Thus, this objective's value is 1 for any environment and any policy. In other words, for any environment, all policies are equally optimal. Therefore, the objective is trivially PAC-learnable. However, due to discontinuity at $\hat{w}$, the objective is neither uniformly continuous nor computable.

The second example is an objective that is not computable but not PAC-learnable. This example simply utilizes the fact that any computable objective plus a non-computable constant, such as the Chaitin's constant, is not computable. Yet, addition of a constant does not affect the induced ordering of policies by the objective. Therefore, the non-computable objective is PAC-learnable by simply ignoring the constant.

Although my condition is only sufficient, to my knowledge, no existing objectives in the literature have a similar nature to these examples that make my conditions inapplicable.

5.11 Proof of Computability of Listing 5.6

Let $g_{a:b}(w_\epsilon, w) \triangleq \sum_{i=a}^{b} R(u_i) \prod_{j=1}^{i-1} \Gamma(u_j)$ denote the partial sum of g in Equation (5.2) from a to b. Let z_h be the sequence of ϵ-moves of length-h that maximizes $g_{0:h}$, that is: $z_h \triangleq \arg\max_{\hat{w_\epsilon} \in \mathcal{E}^h} g_{0:h}(\hat{w_\epsilon}, w)$. Similarly, let $z_\infty \in \mathcal{E}^\omega$ be the infinite-length path of ϵ-moves that maximizes g. Then I can write Δ as:

$$\begin{aligned}
\Delta &\triangleq |[\![(L, \gamma_1, \gamma_2)]\!](w) - \texttt{bozkurt_objective}(w, n)| \\
&= \left| \max_{w_\epsilon \in \mathcal{E}^\omega} g(w_\epsilon, w) - \max_{\hat{w}_\epsilon \in \mathcal{E}^H} g_H(\hat{w}_\epsilon, w) \right| \\
&= |g(z_\infty, w) - g_{0:H}(z_H, w)|
\end{aligned}$$

Observe that $g(z_\infty, w) \geq g_{0:H}(z_H, w)$. To see this inequality, let $\tilde{w}_\epsilon \in \mathcal{E}^\omega$ be any infinite-length path of ϵ-moves with z_H as the prefix. Then I must have $g(\tilde{w}_\epsilon, w) \leq g(z_\infty, w)$, since z_∞ maximizes g. Moreover, since $g_{0:H}(z_H, w)$ is just a partial sum of $g(\tilde{w}_\epsilon, w)$ and since each term of the summation is positive (because R and Γ are postive), I have $g_{0:H}(z_H, w) \leq g(\tilde{w}_\epsilon, w)$. I chain the inequalities to get $g(z_\infty, w) \geq g_{0:H}(z_H, w)$. Therefore I may drop the absolute value: $\Delta = g(z_\infty, w) - g_{0:H}(z_H, w)$.

I now bound Δ by bounding $g(z_\infty, w)$:

$$\begin{aligned}
\Delta &= g(z_\infty, w) - g_{0:H}(z_H, w) \\
&= g_{0:H}(z_\infty, w) + g_{H:\infty}(z_\infty, w) - g_{0:H}(z_H, w) \\
&\leq g_{H:\infty}(z_\infty, w)
\end{aligned}$$

The second equality holds by splitting the summation in g. The last inequality holds since z_H maximizes $g_{0:H}$: $g_{0:H}(z_H, w) \geq g_{0:H}(z_\infty, w)$.

Therefore, after expanding the definition of $g_{H:\infty}$, I have $\Delta \leq \sum_{i=H}^{\infty} R(u_i) \prod_{j=1}^{i-1} \Gamma(u_j)$. Since $R(u_i) < 1$ and $\Gamma(u_j) \leq \max(\gamma_1, \gamma_2)$, I have $\Delta \leq \sum_{i=H}^{\infty} \max(\gamma_1, \gamma_2)^{i-1}$. Simplifying the sum, I get $\Delta \leq \frac{\max(\gamma_1, \gamma_2)^{H-1}}{1 - \max(\gamma_1, \gamma_2)}$. Finally, I plug in the value of

$$H = \frac{(\lfloor \log_2(1 - \max(\gamma_1, \gamma_2)) \rfloor - n)}{\lceil \log_2 \max(\gamma_1, \gamma_2) \rceil}$$

and get $\Delta \leq 2^{-n}$. Therefore, the objective is computable and $[\![(L, \gamma_1, \gamma_2)]\!]$-PAC-learnable.

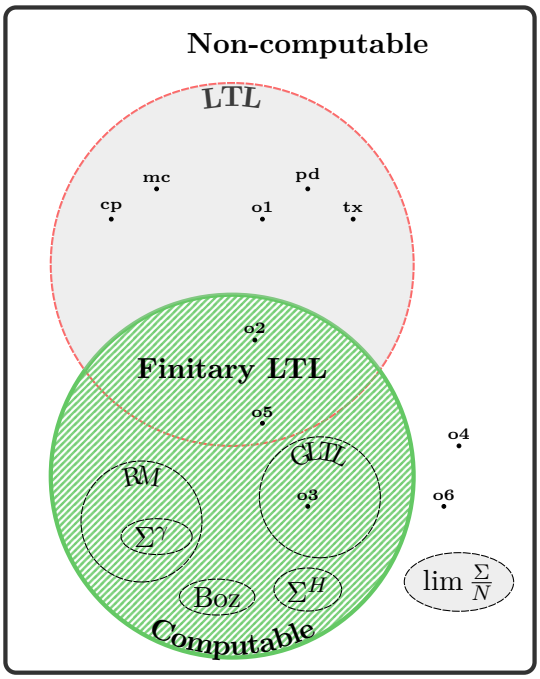


Figure 5.2: Landscape of objectives' learnability up to the current chapter. Dashed circle: LTL objectives; Gray area: objectives provably not PAC-learnable in unrestricted environment; Green slanted area: finitary LTL objectives, PAC-learnable. Empty area: objectives not expressible up to this chapter, and their PAC-learnability is unknown.

5.12 Chapter Summary

Figure 4.5 summarizes the landscape of example objectives' learnability up to the current chapter. In addition to the finitary LTL objectives, including example objectives **o2**, **o5** — which follows from the results in Chapter 4, the following objectives and classes of objectives are provably PAC-learnable:

- Finite-horizon rewards (Σ^H),
- Discounted rewards (Σ^γ),
- Reward Machine (RM) objectives,
- The LTL surrogate objective proposed by Bozkurt et al. [1] (Boz),
- Generalized Linear Temporal Logic (GLTL) objectives.

In contrast, two classes objectives, namely limit-average reward objective $\lim \frac{\Sigma}{N}$ and infinite-horizon LTL objectives, are provably not PAC-learnable in unrestricted environments. These results underscore the limitations of the classical PAC-learnability framework for certain objectives and motivate the need for alternative approaches, such as limit PAC-learnability, to analyze more complex and non-computable objectives.

This work studies the PAC-learnability of general reinforcement-learning objectives and gives the first sufficient condition of PAC-learnability of an objective. Figure 5.1 summarizes my main result. I use examples to show the applicability of my condition on various existing objectives whose learnability were previously unknown.

Applications to Existing Objectives Although I only demonstrated three examples, my theorem also applies to other objectives in the literature. Some examples are (1) modifications to the simple reward machine such as the (standard) reward machine [23] (where rewards depend on not only the reward machine's state but also the environment's state) and the stochastic reward machine [54], (2) other LTL surrogate objectives [2, 16, 17], and (3) various finite-horizon objectives [18, 24, 25].

Moreover, I gave example objectives in Section 5.10 showing my condition is sufficient but not necessary. However, to my knowledge, no previous objective has a similar pattern to my examples. Therefore, I conjecture that my condition applies to most, if not all, existing PAC-learnable objectives in the literature. Nonetheless, verifying each objective's PAC-learnability is out of scope of this work.

Guiding The Design of New Objectives My main result could also help the design of new objectives. With the sufficient condition, researchers can create continuous and computable objectives by design, and the condition will ensure their PAC-learnability.

```
def rl_general_objective(epsilon, delta, objective, mdp, label_fn):
  epsilon_prime = epsilon / 2
  H = modulus_of_continuity(objective, epsilon_prime)

  def lifted_transition(state, action):
    # Current MDP state is the last state in the history
    if len(state) == 1:
      mdp_state = state[-1]
    else:
      last_action, mdp_state = state[-1]
    # Step to sample the environment
    next_mdp_state = mdp.step(mdp_state, action)
    # Append next MDP state
    next_state = state + ((action, next_mdp_state),)
    return next_state

  def reward_fn(state, action):
    if len(state) == H:
      # If current state is the last state in the horizon H,
      # then give reward of the value of the approximated objective
      return objective(state, epsilon_prime)
    else:
      return 0  # Reward 0 otherwise

  # Invoke existing PAC reinforcement-learning algorithm
  # for finite-horizon cumulative rewards
  policy = rl_finite_horizon_cumulative_rewards(
    epsilon_prime,
    delta,
    horizon=H,
    mdp=MDP(
      step=lifted_transition,
      reward_fn=reward_fn,
      init_state=(mdp.init_state,)
    )
  )

  return policy
```

Listing 5.9: Pseudocode for a reinforcement-learning algorithm for computable objectives

Chapter 6

Non-Computable Objectives

As established in Chapter 5, computable objectives are learnable within the PAC-MDP framework and widely applicable in practice. However, many reinforcement learning objectives encountered in research and applications are non-computable and, under classical definitions, not PAC-learnable. Notable examples include the limit-average reward [33] and infinite-horizon LTL objectives [22], as discussed in Chapter 4. Among the gridworld objectives in Chapter 1, which I repeat in Table 6.1 below for convenience, the objectives **o1**, **o5**, and **o6** are all non-computable, and no existing policy learning approach provides theoretical guarantees for these objectives.

This chapter presents my contributions toward addressing the challenges of representing and learning policies for non-computable objectives.

6.1 Overview

I informally illustrate my approach to non-computable objectives using the gridworld examples **o5** and **o6**. I formally analyze these examples in Section 6.4, but here, I introduce them intuitively to provide motivation and an overview of my approach.

Example Gridworld Objectives	
o1	Reach goal without stepping on water
o2	Do **o1** within $n = 15$ steps
o3	Do **o1** within $n \sim \text{Geom}(\frac{1}{15})$ steps
o4	Do **o1**, then retrace the steps back
o5	Do **o4** within $n = 30$ steps
o6	Repeat **o4** forever

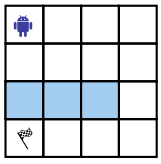


Table 6.1: Example gridworld objectives repeated from Table 1.2.

Following the illustrations, I discuss prior work on non-computable objectives to contextualize the challenges and further justify the need for a general, unified framework for representing and learning non-computable objectives.

6.1.1 Examples

The following gridworld objectives from Table 6.1 illustrate the core challenges of non-computable objectives:

- The objective **o1** requires the agent to reach the goal state while avoiding water.
- The objective **o5** requires the agent to perform the objective **o1** and then retrace its path back to the initial state.
- The objective **o6** requires the agent to repeat the process of the objective **o5** forever.

The objective **o1** is a simpler case of **o5** — I omit its informal discussion here for brevity. I formally analyze all three and more examples in Section 6.4.

Below, I illustrate my approach to the representation and learnability of **o5** and **o6**. By *representation*, I mean a formal language in which every objective has at least one finite-length string in that language that denotes that objective. This definition aligns with the standard definition of representation in computability theory, which I reviewed in Section 5.2. A formal representation is essential for formally defining objectives and analyzing objectives' properties rigorously, particularly their learnability.

6.1.1.1 Representation

As discussed in Chapter 4, the objective **o1** is an infinite-horizon LTL objective: even though it is representable using the formal language LTL, it is not computable. On the other hand, the objective **o5** is not computable nor representable in LTL. In this chapter, I introduce a formal representation for **o1**, **o5**, and, more broadly, all non-computable objectives.

Specifically, using results from mathematical analysis and computability theory, I establish the first general formal representation of non-computable objectives. Specifically, I show that these objectives are expressible as nested limits of computable functions. The non-computability arises from the limits, while the computable function's program text fully encodes the objective. This representation is universal, encompassing all definable objectives without relying on non-computable constants.

I separate the representation into two cases: single-limit and multi-limit. The single-limit case includes objectives like **o1** and **o5**, and is a simpler, special case of the multi-limit case that enjoys a simpler condition for their learnability, as I will discuss later.

```
def reach_and_retrace_helper(w: List[State]) -> Boolean:
  """Check if the target is reached and retracted from the reversed path
  within the finite-length trajectory w.
  """
  history = []
  for i, s in enumerate(w):
    if s is target:
      break
    history.append(s)
  if len(history) > len(w) - i:
    return False
  for s1, s2 in zip(w[i:], reversed(history)):
    if s1 != s2:
      return False
  return True

def reach_and_retrace(n: Natural, w: Iterator[State]) -> Real:
  return real(reach_and_retrace_helper(list(islice(w, n))))
```

Listing 6.1: Pseudocode for the objective **o5**. `real` casts a Boolean to a real number; `islice(w, n)` slices an iterator w to the first n elements.

Single-limit Case Consider the objective **o5**. A computable function, given in Listing 6.1, checks whether the agent successfully completes **o1** and then retraces its path back to the starting state within a finite horizon of$n$steps. Formally, the objective **o5** is defined as the limit of this function as$n$approaches infinity:

$$\lim_{n\to\infty} \texttt{reach\_and\_retrace}(n, w)$$

This limiting process captures the non-computability of **o5**: While the function is computable for any fixed n, performing the check in the limit of infinite n is not computable.

Multi-limit Case The objective **o6** introduces an additional limit. Instead of verifying a single "reach the goal while avoiding water, then retrace the steps back", it requires the agent to satisfy the check forever. To represent the objective formally, I express it as a nested limits of the computable function in Listing 6.2.

Specifically, the function in Listing 6.2 checks whether the agent reaches the goal while avoiding water, then retraces its path within n_2 steps. It does this iteratively, starting from the first state of the trajectory, the second state, and so on, up to the n_1-th state. The

```python
def reach_and_retraces(n2: Natural, w: Iterator[State]) -> Iterator[Boolean]:
  w = list(islice(x, n2))
  for i in range(n2):
    yield reach_and_retrace_helper(w[i:])
  yield from repeat(False)

def always0(n: Natural, w: Iterator[Boolean]) -> Boolean:
  """Check if all input elements in the stream is True, up to n elements."""
  return all(islice(w, n))

def always_reach_and_retrace(n1: Natural, n2: Natural, w: Iterator[State]) -> Boolean:
  return always0(n1, reach_and_retraces(n2, w))
```

Listing 6.2: Pseudocode for the objective **o6**.

objective **o6** is then defined by first taking n_2 to infinity, followed by n_1:

$$\lim_{n_1 \to \infty} \lim_{n_2 \to \infty} \texttt{always\_reach\_and\_retrace}(n_1, n_2, w)$$

The nested limits in this definition account for the non-computability of **o6**.

6.1.1.2 Learnability

I introduce *limit PAC-learnability*, a weaker learnability criterion than the classical PAC-MDP framework, designed for non-computable objectives.

Informally, limit PAC-learnability requires approximating a non-computable objective in probability by a sequence of PAC-learnable objectives in the limit. Objectives satisfying this criterion exhibit a desirable property: within a finite or, more generally, a compact policy set, the near-optimal policies for the PAC-learnable objectives eventually become near-optimal for the non-computable objective.

This property suggests an executable policy-learning strategy with an in-the-limit performance guarantee of near-optimality:

- Learn near-optimal policies for a sequence of PAC-learnable objectives that approximate the non-computable objective in probability in the limit.
- At some unknown but finite step, the learned near-optimal policies for the approximations also become near-optimal for the true non-computable objective.

This guarantee is weaker than the classical PAC-MDP guarantee because it does not ensure near-optimality within a finite number of samples or computations. However, it remains

valuable for the following reasons: When the guarantee holds, the learned policy optimizes the correct objective in the limit of infinite samples and computation. In other words, the guarantee ensures that after consuming sufficient resources, the learning process identifies a near-optimal policy for the non-computable objective. Conversely, if the guarantee fails for an objective, the learned policy may bear no meaningful relation to the optimal policy, meaning the learning algorithm is optimizing for the wrong objective — a phenomenon that plagues in practice, as discussed in Section 1.1.3.

For the above strategy to be executable, each objective in the sequence must be PAC-learnable under the given finite policy set. That is, there must exist an algorithm that identifies a near-optimal policy from the policy class with high probability. For computable objectives, such algorithms exist in the planning-with-generative-model setting under a fixed, finite set of deterministic policies [39]. Therefore, the problem reduces to policy learning for a sequence of computable objectives, so that in the limit, the learned policies eventually become near-optimal for the non-computable objective. Morover, assuming access to such algorithms, the results in this chapter extend to compact policy classes (e.g., stochastic finite-memory policies) and the reinforcement learning setting, providing a concrete strategy for learning near-optimal policies for non-computable objectives in the limit.

Furthermore, I provide a sufficient condition that simplifies proving limit PAC-learnability for non-computable objectives. This condition decomposes the proof into a set of convergence conditions on the individual components of the objective, where each component is expressed as a single limit of a computable function. The condition imposes constraints on the rate at which each computable function converges to its limit, relative to the rates of outer components. For single-limit objectives, this condition is trivially satisfied, leading to the corollary that any non-computable objective defined as a single limit of a computable function is limit PAC-learnable.

Both example objectives satisfy limit PAC-learnability, as I illustrate below.

Single-Limit Case Recall that **o5** is formally a single limit of a computable function. Limit PAC-learnability implies the existence of a sequence of PAC-learnable objectives such that learning a near-optimal policy for each objective in the sequence eventually yields a near-optimal policy for the non-computable **o5**. This sequence is precisely the sequence of computable functions defined in Listing 6.1, obtained by varying the input cutoff n. Using the sufficient condition from this chapter, a single limit of a computable function is trivially limit PAC-learnable. Specifically, a strategy for learning a policy for this objective is:

- Fix a finite set of deterministic policies of interest, for example, a set of deterministic policies with a fixed amount of memory.

- Learn a near-optimal policy for the computable objective **o5** with cutoff $n = 1$.
- Learn a near-optimal policy for the computable objective **o5** with cutoff $n = 2$.
- Continue this process for $n = 3, 4, \ldots$
- Eventually, at some unknown but finite n, the learned near-optimal policies become near-optimal for the true non-computable **o5** within the fixed policy set.

For each learning step, we require a PAC-MDP algorithm that learns a near-optimal policy for the computable objective within the fixed policy set. Such an algorithm exists for the planning-with-generative-model setting under a fixed set of deterministic policies [39]. Moreover, assuming access to such algorithms, the results in this chapter extend to compact policy classes (e.g., stochastic finite-memory policies) and the reinforcement learning setting, providing a concrete strategy for learning near-optimal policies for non-computable objectives in the limit.

Multi-Limit Case The objective **o6** is formally a nested limit of a computable function. For **o6**, the sequence of computable objectives is given in Listing 6.2. In order to form a single sequence of computable and PAC-learnable objectives, I parametrize the two input cutoffs, $n_1(n)$ and $n_2(n)$, using a single parameter n, such that both approach infinity as n grows. Furthermore, using the sufficient condition from this chapter, I prove that **o6** is limit PAC-learnable if $n_2(n)$ grows at a faster rate than $n_1(n)$. A concrete strategy for learning a policy for this objective follows:

- Fix a finite set of deterministic policies of interest.
- Choose a parameterization of $n_1(n)$ and $n_2(n)$ satisfying the rate condition, such as $n_1(n) = n$ and $n_2(n) = n^2$.
- Learn a near-optimal policy for the computable objective in Listing 6.2 with $n = 1$, that is, $n_1 = 1$ and $n_2 = 1$.
- Learn a near-optimal policy for the computable objective with $n = 2$, that is, $n_1 = 2$ and $n_2 = 4$.
- Continue this process for $n = 3, 4, \ldots$
- Eventually, at some unknown but finite n, the learned near-optimal policies become near-optimal for the true non-computable **o6** within the fixed policy set.

As in the case of **o5**, for each learning step, I assume access to a PAC-MDP algorithm that learns a near-optimal policy for the computable objective within the fixed policy set.

6.1.2 Prior Work

Existing research on non-computable objectives has primarily focused on specific subclasses, often characterized using formal languages such as LTL. However, a general and unified framework for representing all non-computable objectives remains elusive. Moreover, the classical PAC-learnability criterion is overly restrictive in this setting. To my knowledge, all substantive non-computable objectives — excluding the contrived examples in Section 5.10 — fail to meet this criterion.

Representation Prior works [29, 55], including those discussed in earlier chapters, define general objectives as functions that assign a ranking to each trajectory. However, this definition lacks an explicit *representation* of arbitrary objectives.

While computable objectives are representable by their program text in a Turing-complete language, the explicit representation of arbitrary non-computable objectives remains unclear. Existing research typically focuses on specific subclasses and employs various formal or mathematical languages for representation:

- Limit-average reward objectives are mathematically defined as the limit of the expected average reward [33].
- Infinite-horizon LTL objectives are expressed in Linear Temporal Logic (LTL) [22].
- ω-regular objectives are specified using ω-regular automata [16].

In the related field of planning, where the environment is known, the Planning Domain Definition Language (PDDL) provides a formal framework for describing planning problems [56]. PDDL can specify objectives such as state reachability, which is non-computable. While these languages effectively capture particular classes of non-computable objectives, they do not generalize to all possible objectives.

In practice, reinforcement-learning practitioners often describe objectives informally using natural language [57], which may be non-computable when formalized. As discussed in Section 1.1.3, rather than formalizing these objectives directly, practitioners commonly design reward-based surrogates — such as discounted cumulative rewards — hoping that optimizing the surrogate will yield good performance on the true objective. However, as noted in Chapter 1, this approach is vulnerable to reward hacking, where the learned policy optimizes the surrogate but fails to align with the intended objective. This issue is particularly pronounced for non-computable objectives, as the absence of a formal representation makes it difficult — if not impossible — to analyze the relationship between the surrogate and the true objective.

To my knowledge, no formal representation exists for general non-computable objectives.

Learnability In Section 5.10, I demonstrated that while computable objectives are PAC-learnable under the classical PAC framework, only contrived non-computable objectives satisfy this criterion. Thus, classical PAC-learnability is not a meaningful guarantee for non-computable objectives. Addressing this gap requires a weaker, yet practically relevant, learnability criterion.

Several works [15, 17, 58, 59] focus on empirical performance without providing theoretical guarantees on the optimality of learned policies. Other research, building on Chapter 4, has introduced relaxed PAC guarantees for specific subclasses of non-computable objectives. For example:

- Voloshin et al. [60] assume partial environment knowledge to provide PAC guarantees for LTL objectives.
- Svoboda, Bansal, and Chatterjee [61] establish a relaxed PAC-like guarantee based on the expected finite steps required to reach a target state.

Other approaches propose algorithms for learning policies for non-computable objectives, such as limit-average reward and LTL objectives, often by reducing the problem to learning a policy for a computable surrogate objective (e.g., discounted cumulative rewards). Various works establish different theoretical guarantees for learning non-computable objectives:

- Naik et al. [62] approximate the limit-average reward objective via a sequence of discounted cumulative reward objectives. As the discount factor approaches 1, the learned policy converges to a near-optimal policy for the limit-average reward objective.
- Bozkurt et al. [1] and Shao and Kwiatkowska [63] provide PAC guarantees for learning near-optimal policies for LTL objectives in the limit of a surrogate parameter, as reviewed in Chapter 4.
- Concurrent with this work, Le et al. [64] establish a PAC guarantee for learning near-optimal policies for ω-regular objectives (a superset of LTL objectives) by reducing them to limit-average reward objectives, which can then be optimized in the limit via a sequence of discounted cumulative reward objectives.

These guarantees apply only to specific classes of objectives and do not extend to all non-computable objectives. They are sometimes referred to as *asymptotic guarantees* [27, 33, 64], meaning that optimality is achieved only in the limit — asymptotically — of infinite samples or computation. However, the precise definition of asymptotic guarantees varies across works and objective classes.

The absence of a formal representation for general non-computable objectives has hindered progress in establishing a general criterion for learning such objectives. To address the

learnability challenges of non-computable objectives, it is first necessary to develop a formal representation framework. This foundation would pave the way for a unified criterion for learning non-computable objectives.

6.2 Non-computable Objective

Before discussing the learnability of any objective, computable or not, the first step is to establish a formal representation. For computable objectives laid out in Chapter 5, the representation is straightforward: it is simply the program text of the objective.

However, non-computable objectives require a different approach. Natural language descriptions are inadequate for formal analysis, necessitating a rigorous formalism to represent non-computable objectives. I address this by representing non-computable objectives as a nested limit of computable functions. This universal representation encompasses all non-computable objectives up to a non-computable constant.

Then, the classic notion of PAC-learnability is not sufficient for non-computable objectives: As I have shown in Chapter 4, the infinite-horizon LTL objectives are not PAC-learnable and not computable. Therefore, I introduce the concept of limit PAC-learnability, a learnability criterion suited for non-computable objectives that requires the agent to learn a sequence of objectives approximating the non-computable objective in the limit. The new criterion has the desirable property that, the near-optimal policy for the computable objective becomes near-optimal for the non-computable objective in the limit.

6.2.1 Representation

I represent each objective $\kappa\colon S^\omega \to \mathbb{R}$ as a k-nested limits of a computable objective $\mathsf{\kappa}\colon (\mathbb{N}^k \times S^\omega) \to \mathbb{R}$. Given a Turing-complete programming language with alphabet Σ, let $K \in \Sigma^*$ be a finite-length string in the language whose denotation is the computable function $\mathsf{\kappa}$: $\mathsf{\kappa} = [\![K]\!]$. Then, the objective κ is represented as

$$\kappa(w) = \lim_{n_1 \to \infty} \dots \lim_{n_k \to \infty} \mathsf{\kappa}(n_1, \dots, n_k, w) \tag{6.1}$$

The program text K of $\mathsf{\kappa}$ completely represents the objective κ. In this chapter, I use teletype font symbols such as $\mathsf{\kappa}$, f, and n to indicate that the symbol refers to a computable function.

When $k = 0$, the objective is computable and falls out of the scope of this chapter — I discuss computable objectives in Chapter 5. The current chapter focuses on cases where $k \geq 1$: Due to the nested limits, κ is non-computable in such cases.

Since nested limits appear frequently in the discussion, to avoid the clutter, for a $k+1$ argument function $f\colon (\mathbb{N}^k \times X) \to Y$ whose first k arguments are natural numbers, I use $\lim[f]$ to denote the function $X \to Y$ that takes each of the first k arguments of f to infinity in order:

$$\lim[f](x) \triangleq \lim_{n_1 \to \infty} \dots \lim_{n_k \to \infty} f(n_1, \dots, n_k, x)$$

I therefore write $\kappa = \lim[\kappa]$ to mean that the computable objective κ represents the non-computable objective κ.

Representation is Universal The next theorem states the expressiveness of this representation: It covers all objectives except those that depend on a non-computable constant. In other words, I consider all objectives where non-computability arises from the nested limits of evaluating the objective over the system's trajectory instead of an outside source of non-computability. [1]

Theorem 6.2.1. *Either Equation (6.1) holds or there exists a finite set Σ and a computable function $\kappa\colon (\mathbb{N}^i \times S^\omega \times \Sigma^\omega) \to \mathbb{R}$ and a non-computable constant $C \in \Sigma^\omega$ such that*

$$\kappa(w) = \lim_{n_1 \to \infty} \dots \lim_{n_k \to \infty} \kappa(n_1, \dots n_k, w, C)$$

I prove Theorem 6.2.1 in Section 6.5.1, where I leverage two key lemmas: (1) A Borel-measurable function is a nested limit of a continuous function. (2) A continuous function is a computable function with access to a potentially non-computable constant. The first lemma is a standard result in descriptive set theory Kechris [65, Theorem 11.6]. I prove the second lemma using a theorem due to Yasugi, Mori, and Tsujii [66], which states that computable functions are dense in the space of continuous functions.

6.2.2 Limit PAC-learable

Intuitively, I want to use a sequence of learnable objectives to approximate the non-computable objective, in the sense that by learning each objective in the sequence, the near-optimal policy for the learned objective becomes near-optimal for the non-computable objective in the limit of the sequence. However, the exact index for when the approximation becomes near-optimal is environment-dependent and unknown.

[1] My results are generalizable to the case of oracle access to the non-computable constant. But, due to a lack of practical objectives that depend on such non-computable constants and simplicity, I assume the representation in Equation (6.1) holds.

Definition 6.2.2 (Limit PAC-learnability)**.** *Let* Π *be a finite set of deterministic policies for the MDP* $\mathcal{M}$*. An objective* κ *is* limit PAC-learnable *if there exists a sequence of PAC learnable objectives* $\{\kappa_n\}_{n=0}^{\infty}$ *such that for all Markov chain* $\mathcal{D}$ *induced by the MDP* $\mathcal{M}$ *and the policy* $\pi \in \Pi$*,* κ_n *converges to* κ *in probability. That is, for all* $\epsilon > 0$*,*

$$\lim_{n\to\infty} \mathsf{P}_{w\sim\mathcal{D}}(|\kappa_n(w) - \kappa(w)| > \epsilon) = 0$$

The sequence $\{\kappa_n\}_{n=0}^{\infty}$ *is a* limit PAC-learnable realization*, or simply, a* realization*, for* κ*.*

Limit PAC-learnability implies that, in practice, as the agent learns near-optimal policies for the sequence of learnable objectives one by one, at some unknown but finite index n, the learned near-optimal policy for approximation κ_n is also near-optimal for the true, non-computable objective κ. The following proposition formalizes this intuition.

Proposition 6.2.3. *Let* κ *be a non-computable objective and* Π *be a finite set of deterministic policies for the MDP* $\mathcal{M}$*. If* κ_n *is a limit PAC-learnable realization for* κ*, then there exists* n *such that the optimal policy for* κ_n *is also optimal for* κ *in the limit:*

$$\lim_{n\to\infty} \max_{\pi\in\Pi} \mathsf{E}_{w\sim\mathcal{D}}[\kappa_n(w)] = \max_{\pi\in\Pi} \mathsf{E}_{w\sim\mathcal{D}}[\kappa(w)]$$

Proof. Since κ_n and κ are bounded, convergence in expectation is equivalent to convergence in probability:

$$\lim_{n\to\infty} \mathsf{E}_{w\sim\mathcal{D}}[\kappa_n(w)] = \mathsf{E}_{w\sim\mathcal{D}}[\kappa(w)]$$

Therefore, take max over the deterministic policies Π on both sides:

$$\max_{\pi\in\Pi} \lim_{n\to\infty} \mathsf{E}_{w\sim\mathcal{D}}[\kappa_n(w)] = \max_{\pi\in\Pi} \mathsf{E}_{w\sim\mathcal{D}}[\kappa(w)]$$

Since Π is finite, the limit and the max are interchangeable. This completes the proof. □

A key requirement for the execution of the above practical strategy is that each objective in the sequence is PAC-learnable under the given finite set of policies, meaning a practioner can run an algorithm to identify a near-optimal policy from the given policy class with high probability. In this chapter, I assume the availability of such an algorithm for a computable objective. Such an algorithm is available for computable objectives in the planning-with-generative-model setting under a fixed finite set of deterministic policies. In particular, [39] provides such an algorithm for the case of planning-with-generative-model setting under a fixed finite set of deterministic policies, such as finite-memory deterministic policies for finite-horizon cumulative reward objectives. Then, using the result laid out in Chapter 5, the

algorithm is extendable to computable objectives. The above definition and proposition, and their derived theorems in the rest of this chapter, are generalizable to the case of compact policy classes, such as stochastic finite-memory policies or the reinforcement learning setting. However, I do not discuss the generalization here, since I am not aware of PAC-learning reinforcement-learning or planning-with-a-generative-model algorithms that identify a near-optimal policy for computable objectives in an arbitrary compact policy class. Nonetheless, provided the availability of such algorithms, the results in this chapter is extendable to the case of compact policy classes and gives rise to a practical strategy to learn near-optimal policies for non-computable objectives in those settings.

6.3 Condition for Limit PAC-learnability

This section establishes a sufficient condition for the limit PAC-learnability of a non-computable objective κ. Proving limit PAC-learnability directly on a non-computable is challenging due to the nested limits of $\lim[\kappa]$. The key idea of the sufficient condition that addresses this challenge is to break down the global convergence condition on κ into a series of convergence conditions on the components of the nested composition of κ, where each component is a single limit of a computable function. Each convergence condition imposes a constraint on the choice of the index to the component. In particular, each constraint requires the index of a component to grow at a rate fast enough relative to the indices of the outer components. More specifically, it requires the growth rate to be large enough so that the computable function at the current layer converges to its limit in probability in the face of the compounding error induced by outer layers.

In this section, I formalize these insights and present the main theorem. I first introduce the decomposition of κ into nested components. I then present the sufficient condition for limit PAC-learnability that leverages the decomposition.

6.3.1 Decomposition of Non-computable Objective

To establish limit PAC-learnability of an objective $\kappa = \lim[\mathsf{\kappa}]$, I first decompose it as a composition of non-computable functions, $\lim[\mathsf{f}_i] \colon X_i \to X_{i-1}$, each a single limit of a computable function $\mathsf{f}_i \colon (\mathbb{N} \times X_i) \to X_{i-1}$.

$$\kappa(w) = \left(\lim[\mathsf{f}_1] \circ \ldots \circ \lim[\mathsf{f}_k]\right)(w) \tag{6.2}$$

where each X_i is a computably compact metric space with the metric d_i, $X_k = (S \times A)^\omega$ and $X_0 = \mathbb{R}$.

The following proposition asserts that such a decomposition is not unique and always possible for any non-computable objective.

Proposition 6.3.1. *Any objective* $\lim[\mathsf{\kappa}]$ *decomposes as Equation (6.2).*

I give a constructive proof of this proposition in Section 6.5.2. Although a constructive decomposition exists by this proof, the decomposition might not help prove limit PAC-learnability. In practice, one might need to find a suitable decomposition that makes proving limit PAC-learnability easier.

6.3.2 Sufficient Condition for Limit PAC-learnability

Before stating the main theorem, I recall the definition of the modulus of continuity, and give some notations necessary to state the theorem.

Recall that since $\mathsf{f}_i(n_i, .)$ is computable and X_i is compact, $\mathsf{f}_i(n_i, .)$ is uniformly continuous: For all $\epsilon > 0$, there exists $\delta > 0$ such that for all $x_1, x_2 \in X_i$,

$$|x_1, x_2| \leq \delta \Rightarrow |\mathsf{f}_i(n_i, x_1), \mathsf{f}_i(n_i, x_2)| \leq \epsilon$$

Let $\delta_i^*(n_i, \epsilon)$ be a modulus of continuity of $\mathsf{f}_i(n_i, .)$. Not that, as mentioned in Chapter 5, $\delta_i^*(n_i, \epsilon)$ is a computable function Weihrauch [51, Theorem 6.2.7].

Moreover, Let $F_i = \lim[\mathsf{f}_i] \circ \ldots \circ \lim[\mathsf{f}_k]$ denote the intermediate composition from i to k. Let $\mathsf{n}_1 \ldots \mathsf{n}_k$ be a sequence of computable functions parametrize by n, and define the computable objective $\mathsf{\kappa}_n$ as the composition of the computable functions f_i indexed at n_i:

$$\mathsf{\kappa}_n(w) = \mathsf{f}_1\left(\mathsf{n}_1(n), \mathsf{f}_2\left(\mathsf{n}_2(n), \ldots \mathsf{f}_k\left(\mathsf{n}_k(n), w\right)\right)\right).$$

I now state the main theorem.

Theorem 6.3.2. *Let* $\kappa = \lim[\mathsf{\kappa}]$ *be an objective with the decomposition in Equation (6.2). The objective* κ *is limit PAC-learnable by the realization* $\mathsf{\kappa}_n$ *if for each* $i \in \{1 \ldots k\}$ *the following condition holds: for all* $\epsilon > 0$

$$\lim_{n \to \infty} \mathsf{P}\left(|\mathsf{f}_i\left(\mathsf{n}_i(n), F_{i+1}\right), \lim[\mathsf{f}_i]\left(F_{i+1}\right)| > \epsilon_i(n, \epsilon)\right) = 0 \tag{6.3}$$

where ϵ_i *is recursively defined as:*

$$\epsilon_i(n, \epsilon) = \begin{cases} \epsilon & \text{if } i = 1 \\ \delta_{i-1}^*(\mathsf{n}_{i-1}(n), \epsilon_{i-1}(n, \epsilon)) & \text{otherwise} \end{cases}$$

Here, each condition in Equation (6.3) imposes a constraint on the growth rate of the index $\mathsf{n}_i(n)$ relative to the growth rate of the indices $\mathsf{n}_j(n)$ for the outer components $j < i$. Each constraint is controlled by compounding of the modulus of continuity δ_j^* of all outer components j, recursively up to the desired error ϵ in the definition of convergence in probability. I interpret this theorem separately under the single-limit and multi-limit cases.

Single Limit Case When the objective is a single limit of a computable function, that is, when $k = 1$, the above condition is trivially satisfied by setting $\mathsf{n}_1(n) = n$.

Corollary 6.3.3. *Any objective* $\lim[\kappa]$ *that is a single limit of a computable function* κ *is limit PAC-learnable with the realization* $\kappa_n(.) = \kappa(n, .)$.

To prove this, note that $\epsilon_1(.) = \epsilon$, and the conditionly simply requires convergence of $\mathsf{f}_1(n, .)$ to $\lim[\mathsf{f}_1](.)$ in probability. Since $\mathsf{f}_1(n, .)$ converges to $\lim[\mathsf{f}_1](.)$ in the limit, as in sure-convergence, convergence in probability is trivially satisfied when $\mathsf{n}_1(n) = n$.

I demonstrate the utility of this corollary through examples in Section 6.4.

Multi-Limit Case When the objective κ is a multi-limit of computable functions (i.e., $k > 1$) the theorem simplifies the proof of the limit PAC-learnability of the objective — a condition imposed on the entire objective — into k separate convergence conditions on the components in the objective decomposition. Moreover, the i-th condition imposes a constraint on the growth rate of the index $\mathsf{n}_i(n)$ relative to the growth rate of the indices $\mathsf{n}_j(n)$ for the outer components $j < i$ through a chain of modulus of continuity δ_j^* up to the outermost component. Since the modulus of continuity is a known computable function, the proof required to show the limit PAC-learnability of κ reduces to finding an expression that upper-bounds the probability in Equation (6.3), and showing that this upper bound converges to 0 as n approaches infinity with a suitable choice of $\mathsf{n}_i(n)$. I demonstrate the utility of this theorem with examples in Section 6.4.

6.4 Examples

In this section, I give various examples of non-computable objectives using the representation in Equation (6.1), and show that these objectives are limit PAC-learnable using Corollary 6.3.3 and Theorem 6.3.2 for the single-limit and multi-limit cases, respectively.

```
def real(x: Union[Rational, Natural, Boolean]) -> Real:
  """Casts a finitely representable object to the corresponding real number under
  fastest-converging Cauchy sequence representation."""
  return repeat(x)

def mean(w: List[Rational]) -> Rational:
  return sum(w) / len(w)

def limit_average_reward_cesaro(n: Natural, w: Iterator[State]) -> Real:
  w = [reward(s) for s in islice(w, n)]
  inner_means = []
  for i in range(n):
    means.append(mean(w[:i+1]))
  outer_mean = mean(inner_means)
  return real(outer_mean)

def limit_average_reward_discount(n: Natural, w: Iterator[State]) -> Real:
  gamma = 1 / (n + 1)
  return (1 - gamma) * discounted_sum(map(reward, w), gamma) # real multiplication
```

Listing 6.3: Pseudocode for the Limit Average Reward objective

6.4.1 Single Limit Examples

I give examples of non-computable objectives that are single limits of computable functions. For each objective, I briefly describe the objective and give the pseudocode for the computable function that represents the objective. All these objectives are limit PAC-learnable by the corollary to Theorem 6.3.2.

6.4.1.1 Limit Average Reward

A classic reward-based objective in reinforcement learning is the limit average reward. Conventionally, the limit average reward objective is the average reward under the stationary distribution of the Markov chain induced by the environment MDP and the policy. A common definition, as reviewed in Chapter 2, is the limit average of the expected $T-$step reward as T approaches infinity:

$$\lim_{T\to\infty} \frac{1}{T} \mathsf{E}_{\mathcal{D}}\left[\sum_{i=0}^{T-1} R(w[i])\right].$$

Notably, in this formulation, the limit average reward objective is not a function of the trajectory but a function of the Markov chain. It is not a simple limit average of the reward function (i.e., $\lim_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n-1} R(w[i])$), since this limit might not exist for periodic Markov

chains [32].

Cesàro Mean Representation Instead, one standard alternative formulation [32, Chapter 8.2] of the limit average reward is the Cesàro mean of the rewards:

$$\kappa_{\text{avg}}^{\text{reward}}(w) = \lim_{n\to\infty} \frac{1}{n} \sum_{m=1}^{n} \left(\frac{1}{m} \sum_{i=1}^{m} R(w[i]) \right).$$

For finite state Markov chains, this Cesàro mean always exists and is equivalent to the conventional limit average reward [32]. Listing 6.3 gives the pseudocode for limit average reward under the Cesàro mean formulation.

Note that the `limit_average_reward_cesaro` function calculates the Cesàro mean, `outer_mean`, as a rational number. To be consistent with the general objectives framework where objectives map trajectories to the reals, the last step of `limit_average_reward_cesaro` casts the result to a real number in the fastest-converging Cauchy sequence representation using the function `real(.)` that repeats the rational number indefinitely. This pattern of casting a rational or Boolean to the corresponding real number holds for various examples in this section.

Discounting-based Representation Another standard alternative formulation is the $1-\gamma$ weighted, γ-discounted sum of the rewards:

$$\kappa_{\text{avg}}^{\text{reward}}(w) = \lim_{n\to\infty} (1-\gamma(n)) \sum_{i=0}^{\infty} \gamma(n)^i R(w[i])$$

Here, $\gamma(.)$ is a sequence of discount factors that approaches 0 as n approaches infinity such as $\gamma(n) = 1/(n+1)$. This formulation is equivalent to the conventional limit average reward [62, 67] for finite-state Markov chains. Listing 6.3 gives the pseudocode for limit-average reward under the discounting-based formulations.

Learnability By Corollary 6.3.3, since the limit average reward is a single limit of a computable function in either the Cesàro mean or the discounting-based formulation, the limit average reward objective is limit PAC-learnable with either realization `limit_average_reward_cesaro` and `limit_average_reward_discount` given in Listing 6.3.

6.4.1.2 Reward Lower Bound

The reward lower bound objective maximizes the lower bound of the encountered — un-summed — rewards in the trajectory. In particular, given a bounded reward function

```
def reward_lower_bound(n: Natural, w: Iterator[State]) -> Real:
 w = [reward(s) for s in islice(w, n))]
 return real(min(w))
```

Listing 6.4: Pseudocode for the Reward Lower Bound objective

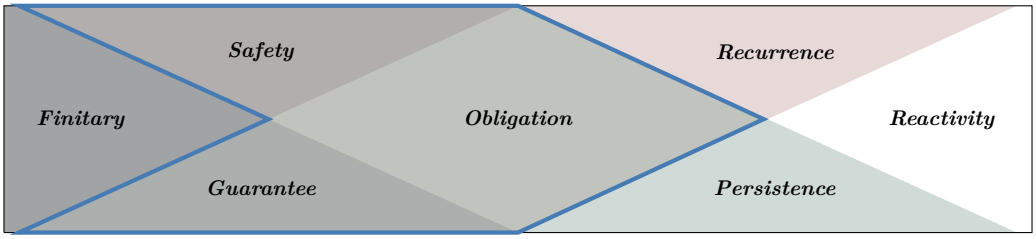


Figure 6.1: Region highlighting non-finitary LTL objectives below the obligation class

mapping from each state $r\colon S \to \mathbb{Q}$, the reward lower bound objective $\kappa\colon S^\omega \to \mathbb{R}$ is:

$$\kappa_{\text{lb}}(w) = \min_{s \in w} r(s)$$

Representation Listing 6.4 gives the pseudocode for the reward lower bound objective. The function `reward_lower_bound` calculates the minimum reward in the trajectory up to the n-th state, and casts the result to a real number. In the limit of n, `reward_lower_bound` becomes exactly the reward lower bound objective.

$$\lim_{n\to\infty} \texttt{reward\_lower\_bound}(n, w) = \lim_{n\to\infty} \min_{s \in w[:n]} r(s) = \min_{s \in w} r(s)$$

The limit exists and equals the reward lower bound objective since the reward function r is a bounded bounded function over a finite the state space.

Learnability By Corollary 6.3.3, since the reward lower bound is a single limit of a computable function, the reward lower bound objective is limit PAC-learnable with the realization `reward_lower_bound` given in Listing 6.4.

6.4.1.3 Non-finitary LTL Formulas Lower Than the Obligation Class

The class of non-finitary LTL objectives below the obligation class, as shown in Figure 6.1, are expressible as a single limit of a sequence of computable functions [43].

```
def eventually0(n: Natural, w: Iterator[Boolean]) -> Boolean:
  """Check if any input element in the stream is True, up to n elements."""
  return any(islice(w, n))

def always0(n: Natural, w: Iterator[Boolean]) -> Boolean:
  """Check if all input elements in the stream is True, up to n elements."""
  return all(islice(w, n))

def build_ltl_single(
  logical_op: Callable[[NTuple[Boolean]], Boolean]) -> Boolean:
  temporal_ops: NTuple[Union[always0, eventually0]],
  dfa: Callable[Iterator[State], Iterator[NTuple[Boolean]]]):
  """Construct a single limit of a sequence of computable functions for
  non-finitary LTL objectives below the obligation class."""

  def ltl_single(n: Natural, w: Iterator[State]) -> Real:
    # Apply the DFA to the input trajectory `w`
    dfa_outputs = dfa(w)  # Outputs multiple sequences from the DFA
    # Apply the temporal operation to each sequence
    temporal_results = [temporal_ops(n, seq) for seq in zip(*dfa_outputs)]
    # Combine the temporal results using the logical operation
    return real(logical_op(*temporal_results))

  return ltl_single
```

Listing 6.5: Pseudocode for constructing a single limit of a sequence of computable functions for non-finitary LTL objectives below the obligation class

Representation Specifically, Manna and Pnueli [43] established that any objective in this class is equivalent to a logical combination of the always (G) and eventually (F) operators applied to a finite set of atomic propositions. A deterministic finite automaton (DFA) tracks these atomic propositions. The DFA is a computable function that takes the environment's trajectory as input and outputs a corresponding trajectory of tuples of Booleans that indicates the satisfaction of each atomic proposition over time.

Listing 6.5 gives an explicit construction of the sequence of computable functions that represent any LTL objective in this class, using the higher-order function `build_ltl_single(.)`. In particular, it takes as input

- a computable function that maps a trajectory to a trajectory of n-tuple of satisfaction of the atomic propositions.
- a n-tuple of either the `always0` or `eventually0` functions,
- a logical operation joining the result of the temporal operators,

and returns a computable function `ltl_single(n, w)` that represents the LTL objective in

```python
def not_water_until_goal_past(w: Iterator[State]) -> Iterator[Tuple[Boolean]]:
  """Check if the agent has not stepped on water until it reaches the goal."""
  sat = True
  for s in w:
    sat = sat and (is_goal(s) or not is_water(s))
    yield (sat, )

o1 = build_ltl_single(lambda x: x, (eventually0, ), not_water_until_goal)
```

Listing 6.6: Pseudocode for the objective **o1**, "reach goal without stepping on water".

the limit of `n` approaching infinity.

Here, the function `always0` is a computable function that checks if all the input elements in the stream are True, from 0 to the n-th input. Similarly, the function `eventually0` checks if any of the input elements in the stream is True, from 0 to the n-th input. The suffix `0` in the function names indicates that the functions perform these checks starting from the 0-th input and return a single Boolean value.

Example: Reach Goal Without Stepping on Water As an example, consider the LTL objective **o1** from Table 1.1, "Reach goal without stepping on water", which is exressible as the LTL formula $\neg \mathit{water}\ \mathsf{U}\ \mathit{goal}$ [2]. The formula is equivalent to the LTL formula $\mathsf{F}\, p$, where p is the atomic proposition that is true at time t iff the agent has not stepped on water until it reaches the goal in the past up to time t.

Learnability By Corollary 6.3.3, since any LTL objective below the obligation class is a single limit of a computable function, such an objective is limit PAC-learnable with the realization `ltl_single` in Listing 6.5. This class includes the objectives **o1** in Listing 6.6.

6.4.1.4 Reach and Retrace

The reach and retrace objective checks if the agent reaches given target state, and whether it retracts from the target state in the reversed path. This objective is an extension of the LTL objective "eventually reach the target state", but it requires the agent to retrace from the target state in the reversed path. This additional requirement makes the objective not capturable by LTL, since LTL cannot memorize the path taken to reach the target state [42].

[2] Note that here instead of the formula $\mathsf{F}\, \mathit{goal} \wedge \mathsf{G}\, \mathit{water}$ used in prior chapters, I assume here that we do not care about the agent stepping on water once it reaches the goal for simplicity.

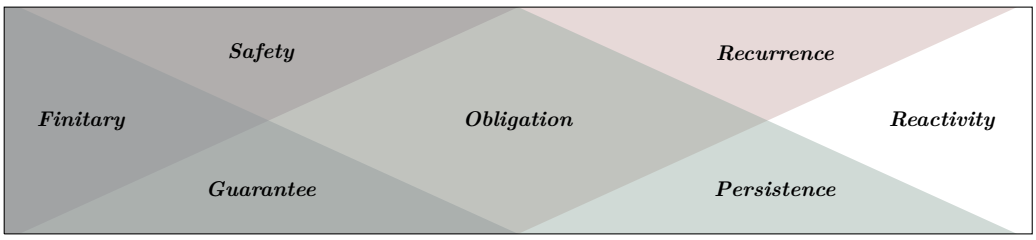


Figure 6.2: Region highlighting LTL objectives above the obligation class

Representation Listing 6.1 gives the pseudocode for the reach and retrace objective. The function `reach_and_retrace_helper` checks if the target is reached and retracted from the reversed path within the finite-length trajectory `w`. The function `reach_and_retrace` slices the first n states of the trajectory `w`, process the trajectory using the function `reach_and_retrace_helper`, then casts the result to a real number in the fastest-converging Cauchy sequence representation.

Learnability By Corollary 6.3.3, since the reach and retrace objective is a single limit of a computable function, the reach and retrace objective is limit PAC-learnable with the realization `reach_and_retrace` given in Listing 6.1.

6.4.2 Multi-Limit Examples

In the multi-limit case, I give examples of non-computable objectives. For each objective, I first give their representation as a nested limit of computable functions. I then prove that they are limit PAC-learnable using the sufficient condition to limit PAC-learnability derived earlier in Section 6.3.

6.4.2.1 Linear Temporal Logic Above the Obligation Class

The class of LTL objectives above the obligation class, shown in Figure 6.2, is expressible as a multi-limit of computable functions. I prove below that they are limit PAC-learnable.

Representation In particular, Manna and Pnueli [43] established that any objective in this class is equivalent to a logical combination of the GF (always eventually) and FG (eventually always) operators applied to a finite set of atomic propositions. A deterministic finite automaton (DFA) tracks these atomic propositions. The DFA is a computable function that takes the environment's trajectory as input and outputs a corresponding trajectory of tuples of Booleans that indicates the satisfaction of each atomic proposition over time.

```
def always(n: Natural, w: Iterator[Boolean]) -> Iterator[Boolean]:
  w = list(islice(w, n))
  for i in range(n):
    yield all(w[i:])
  yield from repeat(False)

def eventually(n: Natural, w: Iterator[Boolean]) -> Iterator[Boolean]:
  w = list(islice(w, n))
  for i in range(n):
    yield any(w[i:])
  yield from repeat(False)

GF = (always0, eventually)
FG = (eventually0, always)

def build_ltl_multi(
  logical_op: Callable[[NTuple[Boolean]], Boolean],
  temporal_ops: NTuple[Union[GF, FG]],
  dfa: Callable[Iterator[State], Iterator[NTuple[Boolean]]]):
  temporal_ops_outer, temporal_ops_inner = zip(*temporal_ops)

  def inner(n: Natural, w: Iterator[State]) -> Iterator[NTuple[Boolean]]:
    return zip(*[op(n, wp) for op, wp in zip(temporal_ops_inner, zip(*dfa(w)))])

  def outer(n: Natural, w: Iterator[NTuple[Boolean]]) -> Boolean:
    return logical_op(*[op(n, wp) for op, wp in zip(temporal_ops_outer, zip(*w))])

  def ltl_multi(n1: Natural, n2: Natural, w: Iterator[State]) -> Real:
    return real(outer(n1, inner(n2, w)))

  return ltl_multi
```

Listing 6.7: Pseudocode for constructing a multi-limit of a sequence of computable functions for LTL objectives above the obligation class

Listing 6.7 gives an explicit construction of the sequence of computable functions that represent any LTL objective in this class, using the higher-order function `build_ltl_multi(.)`. In particular, it takes as input

- a computable function that maps a trajectory to a trajectory of n-tuple of satisfaction of the atomic propositions.
- a n-tuple of pairs of temporal operators, either GF or FG,
- a logical operation joining the result of the temporal operators,

and returns a computable function `ltl_multi(n1, n2, w)` that represents the LTL objective in the limit of `n1` and `n2`.

Learnability The following theorem states that any LTL objective above the obligation class is limit PAC-learnable.

Theorem 6.4.1. *Any LTL objective above the obligation class is limit PAC-learnable with the realization* `ltl_multi`*:*

$$\kappa_n(w) = \texttt{ltl\_multi}(\mathsf{n}_1(n), \mathsf{n}_2(n), w)$$

where $\mathsf{n}_1(n)$ *and* $\mathsf{n}_2(n)$ *are computable functions parametrized by* n*, with the constraints:*

$$\lim_{n\to\infty} \mathsf{n}_1(n) = \infty \quad \textit{and} \quad \lim_{n\to\infty} \frac{\mathsf{n}_2(n)}{\mathsf{n}_1(n)} = \infty$$

The theorem states that a realization of the LTL objectives above the obligation class must satisfy that the index n_2 for the inner temporal operators must grow faster than the index n_1 for the outer temporal operators so that the temporal operators are accurate in the limit of the common index n.

6.4.2.2 Repeating Reach and Retrace

The repeating reach and retrace objective repeats the reach and retrace objective in the single limit case for an infinite number of times. Intuitively, the agent must reach the target state and retract from it in the reversed path. Further, it repeats this same process deterministically an infinite number of times. Listing 6.2 gives the pseudocode for the repeating reach and retrace objective.

I prove that the repeating reach and retrace objective is limit PAC-learnable using the sufficient condition for limit PAC-learnability in Section 6.3.

Representation The objective is formally defined as the double limit of computable function $\mathsf{\kappa}(n_1, n_2, w)$

$$\kappa(w) = \lim_{n_1\to\infty} \lim_{n_2\to\infty} \mathsf{\kappa}(n_1, n_2, w)$$

where the computable function $\mathsf{\kappa}(n_1, n_2, w)$ is the composition of computable functions `always0` and `reach_and_retraces`.

$$\mathsf{\kappa}(n_1, n_2, w) = \texttt{always0}(n_1, \texttt{reach\_and\_retraces}(n_2, w))$$

The function `reach_and_retraces` is a stream variant of the `reach_and_retrace` function in Listing 6.1: For each index of the input infinite-length trajectory, it checks if the target state is reached starting from that index and if the trajectory retraces from the reversed

path to the starting index. In the limit of $n_2 \to \infty$, it produces an infinite stream of Boolean values, where each value is the result of the reach and retrace check starting from the corresponding index.

The function `always0` checks if the input stream is a repeating sequence of `True` values. The limit of $n_1 \to \infty$ produces a single Boolean value, indicating whether the input stream is an infinitely repeating sequence of `True` values.

As a composition of the two functions, the objective evaluates to true if the trajectory repeatedly visits both the initial and target states, using the same path between visits.

Learnability The following theorem states that the repeating reach and retrace objective is limit PAC-learnable.

Theorem 6.4.2. *The repeating reach and retrace objective is limit PAC-learnable by the realization:*

$$\kappa_n(w) = \texttt{always0}(\mathsf{n}_1(n), \texttt{reach\_and\_retraces}(\mathsf{n}_2(n), w))$$

where $\mathsf{n}_1(n)$ *and* $\mathsf{n}_2(n)$ *are computable functions parametrized by* n*, with the constraints:*

$$\lim_{n\to\infty} \mathsf{n}_1(n) = \infty \quad \textit{and} \quad \lim_{n\to\infty} \frac{\mathsf{n}_2(n)}{\mathsf{n}_1(n)} = \infty$$

For example, $\mathsf{n}_1(n) = n$ *and* $\mathsf{n}_2(n) = n^2$ *satisfy the constraints.*

Intuitively, the theorem states that the above realization for the objective must ensure that the index n_2 for the `reach_and_retraces` function grows faster than the index n_1 for the `always0` function, ensuring the reach-and-retrace check remains accurate in the limit of the common index n.

The proof follows from the sufficient condition for limit PAC-learnability given in Section 6.3, and is similar to the proof of limit PAC-learnability for LTL objectives above the obligation class. I give the complete proof in Section 6.5.5.

6.4.2.3 Eventually Repeat Reach and Retrace

The eventual repeat reach and retrace objective is similar to the repeating reach and retrace objective, but instead of requiring the repeat of the reach and retrace process start from the beginning of the trajectory, it requires that the process eventually starts at some point in the trajectory.

```
def eventually_repeat_reach_and_retrace(n1: Natural, n2: Natural, w: Iterator[State]) ->
↪  Real:
  return real(eventually0(n1, always(n2, reach_and_retraces(n3, w))))
```

Listing 6.8: Pseudocode for the Eventually Repeat Reach and Retrace objective

Representation The objective is formally defined as the triple limit of computable function $\kappa(n_1, n_2, n_3, w)$

$$\kappa(w) = \lim_{n_1 \to \infty} \lim_{n_2 \to \infty} \lim_{n_3 \to \infty} \kappa(n_1, n_2, n_3, w)$$

where the computable function $\kappa(n_1, n_2, n_3, w)$ is the composition of computable functions `eventually0`, `always`, and `reach_and_retraces`.

$$\kappa(n_1, n_2, n_3, w) = \texttt{eventually0}(n_1, \texttt{always}(n_2, \texttt{reach\_and\_retraces}(n_3, w)))$$

The function `eventually0` checks if the input stream eventually becomes `True`. In the limit of $n_1 \to \infty$, it produces a single Boolean value, indicating whether the input stream eventually becomes `True`. The function `always` checks if the input stream is a repeating sequence of `True` values. The limit of $n_2 \to \infty$ produces a single Boolean value, indicating whether the input stream is an infinitely repeating sequence of `True` values. The function `reach_and_retraces` is the same as in the repeating reach and retrace objective.

Learnability The following theorem states that the eventual repeat reach and retrace objective is limit PAC-learnable.

Theorem 6.4.3. *The eventually repeating reach and retrace objective is limit PAC-learnable by the realization:*

$$\kappa_n(w) = \texttt{eventually}\left(\mathsf{n}_1(n), \texttt{always}(\mathsf{n}_2(n), \texttt{reach\_and\_retraces}(\mathsf{n}_3(n), w))\right)$$

where $\mathsf{n}_1(n)$ *and* $\mathsf{n}_2(n)$ *are computable functions parametrized by* n*, with the constraints:*

$$\lim_{n \to \infty} \mathsf{n}(n) = \infty, \quad \lim_{n \to \infty} \frac{\mathsf{n}_2(n)}{\mathsf{n}_1(n)} = \infty \quad \textit{and} \quad \lim_{n \to \infty} \frac{\mathsf{n}_3(n)}{\mathsf{n}_2(n)} = \infty$$

For example, $\mathsf{n}_1(n) = n$*,* $\mathsf{n}_2(n) = n^2$ *and* $\mathsf{n}_3(n) = n^3$ *satisfy the constraints.*

The proof is similar to Theorem 6.4.2. I give a sketch here for brevity: In the proof for Theorem 6.4.2, the sufficient condition for limit PAC-learnability given in Section 6.3 expands

to two conditions, corresponding to the limits in $\mathsf{n}_1(n)$ and $\mathsf{n}_2(n)$. Here, the condition expands to three conditions, corresponding to the limits in $\mathsf{n}_1(n)$, $\mathsf{n}_2(n)$, and $\mathsf{n}_3(n)$.

6.5 Proofs

In this section, I give the proofs of the theorems and lemmas presented in this chapter.

6.5.1 Proof of Theorem 6.2.1

To the end of proving Theorem 6.2.1, I first give two key lemmas: (1) A Borel-measurable function is a nested limit of a continuous function. (2) A continuous function is a computable function with access to a potentially non-computable constant. Then, I prove Theorem 6.2.1 using these lemmas.

6.5.1.1 Key Lemmas

The first lemma is a standard result in descriptive set theory:

Lemma 6.5.1 (Theorem 11.6 of Kechris [65], Reworded)**.** *Let X be a metrizable space and $f\colon X \to \mathbb{R}$ a Borel-measurable function. There exists a $(n_1 \dots n_i)$-indexed sequence of continuous functions $\{\hat{f}_{n_1 \dots n_i}\}_{\forall i.n_i=0}^{\infty}$ such that*

$$\forall w.\quad f(w) = \lim_{n_1 \to \infty} \dots \lim_{n_i \to \infty} \hat{f}_{n_1, \dots n_i}(w)$$

The second lemma is a corollary of a theorem due to Yasugi, Mori, and Tsujii [66]:

Theorem 6.5.2 (Theorem 1 of Yasugi, Mori, and Tsujii [66])**.** *Let X be a computably compact metric space and $f\colon X \to \mathbb{R}$ be a continuous function. For any $\epsilon > 0$, there exists a computable function $\mathsf{g}\colon X \to \mathbb{R}$ such thut $\max_{x \in X} |f(x) - \mathsf{g}(x)| < \epsilon$.*

I now state and prove the second lemma:

Lemma 6.5.3. *Let X be a computably compact metric space. Let $f\colon X \to \mathbb{R}$ be a continuous function. There exists an alphabet Σ, a computable function $\mathsf{h}\colon (X \times \Sigma^{\omega}) \to \mathbb{R}$ and a constant $C \in \Sigma^{\omega}$ such that $f(x) = \mathsf{h}(x, C)$.*

Proof of Lemma 6.5.3. Let C be the sequence of descriptions of the computable functions g_{ϵ_i} that approximates f within $\epsilon_i = 2^{-i}$ according to Theorem 6.5.2. Note that since g_{ϵ_i} is a computable function, for any Turing-complete programming language over the alphabet Σ, it has a finite-length description in Σ^{ω}, that is, its program text. Therefore, C is a potentially

non-computable constant. I construct h as an interpreter of the programming language. The interpreter runs the i-th program in C on the input x to produce a (2^{-i})-approximation of $f(x)$. Then, h is computable and $f(x) = h(x, C)$. □

6.5.1.2 Proof of Theorem 6.2.1

Proof of Theorem 6.2.1. If C is a computable constant, then I subsume C into the computable function κ — doing so proves that Equation (6.1) holds. Therefore, it suffices to prove the theorem for the case where C is a potentially non-computable constant.

First, I apply Lemma 6.5.1 to κ:

$$\forall w. \quad \kappa(w) = \lim_{n_1 \to \infty} \dots \lim_{n_i \to \infty} \hat{\kappa}_{n_1,\dots,n_i}(w)$$

It suffices to show that there exists a computable function κ such that

$$\forall w. \quad \hat{\kappa}_{n_1,\dots,n_i}(w) = \kappa(n_1, \dots n_i, w, C)$$

To make notations more concise, the index $(n_1 \dots n_i)$ is enumerable and can be mapped to a single index n. In particular, let $p\colon \mathbb{N}^i \to \mathbb{N}$ be a pairing function, a computable bijection between $\mathbb{N}^i$ and $\mathbb{N}$ such that both p and p^{-1} are computable. To be concrete, I use the generalized Cantor pairing function:

$$p(n_1, \dots, n_i) = p_b(n_1, p_b(n_2, \dots, n_i))$$

where p_b is the binary Cantor pairing function: $p_b(n_1, n_2) = \frac{1}{2}(n_1 + n_2)(n_1 + n_2 + 1) + n_2$. Given indices $n_1, \dots, n_i$, I let $n = p(n_1, \dots, n_i)$ be the corresponding index in $\mathbb{N}$. I can therefore write the $(n_1 \dots n_i)$-indexed sequence $\{\hat{\kappa}_{n_1,\dots,n_i}\}$ as singly indexed sequence $\{\bar{\kappa}_n\}$, where $\bar{\kappa}_n = \hat{\kappa}_{p^{-1}(n)}$.

Since $\bar{\kappa}_n$ is continuous, I apply Lemma 6.5.3 to rewrite it in terms of a computable function κ_n and a constant A_n:

$$\forall w. \quad \bar{\kappa}_n(w) = \kappa_n(w, A_n)$$

I pack the constants $A_n \in \Sigma^\omega$ into a single constant $A \in \Sigma^\omega$ using Cantor's diagonal argument: $A[p_b(n, i)] = A_n[i]$. The function that extracts the C_n from A is computable, and I can subsume it into κ_n:

$$\forall w. \quad \bar{\kappa}_n(w) = \kappa_n(w, A)$$

Now consider the function $K(n)$ that takes in the index n and returns the computable

function κ_n: $K(n) = \kappa_n$. Since each κ_n is computable, I represent it as a program text in Σ^*. Further, I can encode the program texts of all κ_n into a single constant $B \in \Sigma^\omega$, and the function $\tilde{K}$ that extracts the n-th program text from B is computable:

$$K(n) = \tilde{K}(n, B)$$

Finally, I pack the constants A and B into a single constant $C \in \Sigma^\omega$ by alternating between elements of A and B:

$$C[2i] = A[i] \quad \text{and} \quad C[2i+1] = B[i]$$

The functions that extracts A and B from C are computable, denote them as p_A and p_B respectively. Finally defining the computable function κ as

$$\kappa(n_1, \dots n_i, w, C) = \tilde{K}(p(n_1 \dots n_i), p_B(C))(w, p_A(C))$$

completes the proof. □

6.5.2 Proof of Proposition 6.3.1

Proof. In particular, any objective has a trivial decomposition, where each f_k simply packs the arguments n_k and x and outputs the tuple $(\frac{1}{n_k}, x)$ for $i > 1$, and the outermost f_1 unpacks all the tuples and applies the full computation κ:

$$\mathsf{f}_i(n_i, x) = \begin{cases} (\frac{1}{n_i}, *x) & i > 1 \\ \kappa(n_1, \frac{1}{x[1]} \dots \frac{1}{x[i-1]}, x[i]) & i = 1 \end{cases}$$

Note that all domains and codomains of f_is are computably compact metric spaces, and therefore is a valid decomposition as in Equation (6.2). □

6.5.3 Proof of Theorem 6.3.2

Theorem 6.5.4. *The objective $\kappa = \lim[\kappa]$ is limit PAC-learnable and κ_n is a realization for κ if for each $i \in \{1 \dots k\}$, $\lim_{n\to\infty} \mathsf{n}_i(n) = \infty$ and the following condition holds: for all $\epsilon > 0$*

$$\lim_{n\to\infty} \mathsf{P}(|\mathsf{f}_i\left(\mathsf{n}_i(n), F_{i+1}\right), \lim[\mathsf{f}_i]\left(F_{i+1}\right)| > \epsilon_i(n, \epsilon)) = 0$$

where ϵ_i is recursively defined as:

$$\epsilon_i(n, \epsilon) = \begin{cases} \epsilon & \text{if } i = 1 \\ \delta^*_{i-1}(\mathsf{n}_{i-1}(n), \epsilon_{i-1}(n, \epsilon)) & \text{otherwise} \end{cases}$$

Proof. Let F_i be a shorthand for the intermediate composition of the computable functions f_i from i to k:

$$\mathsf{F}_i(n) = \mathsf{f}_i\left(\mathsf{n}_i(n), \ldots \mathsf{f}_k\left(\mathsf{n}_k(n), w\right)\right)$$

I lower bound on the probability that F_i is ϵ_i-close to F_i:

$$\begin{aligned} &\mathsf{P}(|\mathsf{F}_i(n), F_i| > \epsilon_i) \\ &= \mathsf{P}(|\mathsf{f}_i\left(\mathsf{n}_i(n), \mathsf{F}_{i+1}(n)\right), \lim[\mathsf{f}_i]\left(F_{i+1}\right)| > \epsilon_i) \\ &\leq \mathsf{P}(|\mathsf{f}_i\left(\mathsf{n}_i(n), \mathsf{F}_{i+1}\right), \mathsf{f}_i\left(\mathsf{n}_i(n), F_{i+1}\right)| > \frac{1}{2}\epsilon_i) + \mathsf{P}(|\mathsf{f}_i\left(\mathsf{n}_i(n), F_{i+1}\right), \lim[\mathsf{f}_i]\left(F_{i+1}\right)| > \frac{1}{2}\epsilon_i) \end{aligned}$$

where I applied the triangle inequality and the union bound at the last step.

For the first term, since f_i is computable and the domain X_i is compact, it is uniformly continuous. Let δ^*_i be the modulus of continuity of f_i. I upper-bound the first term by

$$\mathsf{P}(|\mathsf{F}_{i+1}(n), F_{i+1}| > \delta^*_i(\mathsf{n}_i(n), \frac{1}{2}\epsilon_i))$$

This bound in turn becomes a condition on $\mathsf{F}_{i+1}(n)$ to be ϵ_{i+1}-close to F_{i+1}, where ϵ_{i+1} is given by the definition in the theorem statement. Therefore, by induction, it suffices to let the second term approach zero in the limit of n. This completes the proof. □

6.5.4 Proof of Theorem 6.4.1

Proof Sketch. First, I pull out the temporal operators combinations ($\mathsf{G}\,\mathsf{F}$ and $\mathsf{F}\,\mathsf{G}$) from the outermost logical operation: If, for $i \in \{1 \ldots k\}$, the sequence of functions $\kappa_{(i),n} \colon S^\omega \to \mathbb{B}$ converges to $\kappa_{(i)} \colon S^\omega \to \mathbb{B}$ in probability, then the function $\kappa_n = \texttt{logical\_op}(\kappa_{1,n}, \ldots, \kappa_{k,n})$ converges to $\kappa = \texttt{logical\_op}(\kappa_{(1)}, \ldots, \kappa_{(k)})$ in probability. Therefore, it suffices to show that for any atomic proposition a, the formulas $\mathsf{G}\,\mathsf{F}\,a$ and $\mathsf{F}\,\mathsf{G}\,a$ are limit PAC-learnable by the realizations $\texttt{always0}(\mathsf{n}_1(n), \texttt{eventually}(\mathsf{n}_2(n), w))$ and $\texttt{eventually0}(\mathsf{n}_1(n), \texttt{always}(\mathsf{n}_2(n), w))$, respectively. Without loss of generality, I focus on the $\mathsf{G}\,\mathsf{F}\,a$ case below.

By the sufficient condition for limit PAC-learnability (Section 6.3), I need to show that the following condition holds for all $\epsilon > 0$:

$$\lim_{n \to \infty} \mathsf{P}(|\texttt{always0}(\mathsf{n}_1(n), w), \lim[\texttt{always0}](w)| > \epsilon) = 0,$$

and

$$\lim_{n\to\infty} \mathsf{P}(|\texttt{eventually}(\mathsf{n}_2(n), w), \lim[\texttt{eventually}](w)| > \delta^*_{\texttt{always0}}(\mathsf{n}_1(n), \epsilon)) = 0,$$

where the the two conditions are due to $i = 1$ and $i = 2$ in the theorem statement, respectively. The first condition is satisfied by letting $\mathsf{n}_1(n)$ grow to infinity:

$$\lim_{n\to\infty} \mathsf{n}_1(n) = \infty$$

so I focus on the second condition.

The function $\delta^*_{\texttt{always0}}$ is the modulus of continuity of the function `always0`. Since the function `always0` inspects only the first n elements of the input stream, and the input stream is the metric space over the infinite-length words $\mathbb{B}^\omega$ which has the metric $d_{\mathbb{B}^\omega}(w_1, w_2) = 2^{-L_{\text{prefix}}(w_1,w_2)}$, where $L_{\text{prefix}}(w_1, w_2)$ is the length of the longest common prefix of w_1 and w_2, it is simply $\delta^*_{\texttt{always0}}(n, \epsilon) = 2^{-n}$. Thus, the condition simplifies to

$$\lim_{n\to\infty} \mathsf{P}(|\texttt{eventually}(\mathsf{n}_2(n), w), \lim[\texttt{eventually}](w)| > 2^{-\mathsf{n}_1(n)}) = 0.$$

Recall the following epsilon-delta definition of the limit of the function `eventually`:

$$\lim[\texttt{eventually}](w) = \lim_{n\to\infty} \texttt{eventually}(n, w)$$

For all $\epsilon > 0$ and $w \in \mathbb{B}^\omega$, there exists n such that for all $n' \geq n$:

$$2^{-L_{\text{prefix}}(\texttt{eventually}(n',w),\lim[\texttt{eventually}](w))} < \epsilon$$

Let $n^*(\epsilon, w)$ be the minimum n that satisfies the above for any given $w \in \mathbb{B}^\omega$, called the *local modulus of convergence* of `eventually` to $\lim[\texttt{eventually}]$. Then I rewrite the above inequaility using the local modulus of convergence:

$$\begin{aligned}
&\mathsf{P}(|\texttt{eventually}(\mathsf{n}_2(n), w), \lim[\texttt{eventually}](w)| > 2^{-\mathsf{n}_1(n)}) \\
&\leq \mathsf{P}(n^*(2^{-\mathsf{n}_1(n)}, w) \geq \mathsf{n}_2(n)) \\
&\leq \frac{\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)]}{\mathsf{n}_2(n)}
\end{aligned}$$

The second step is due to Markov's inequality.

Therefore, by Theorem 6.3.2, it suffices to show that the ratio approaches zero in the limit of n:

$$\lim_{n\to\infty} \frac{\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)]}{\mathsf{n}_2(n)} = 0 \tag{6.4}$$

Here, the numerator is the expected number of steps of the inputs w, so that the first n_1 elements of the output of `eventually` matches that of lim[`eventually`], where the expectation is taken over the distribution of the inputs w induced by the Markov chain. I show that the numerator is $\mathcal{O}(\mathsf{n}_1(n))$.

Consider the i-th output of the function `eventually`, for $i \in \{1 \dots \mathsf{n}_1(n)\}$. Let n_i^* denote the minimum number of inputs `eventually` consumes such that the i-th output matches.

When the input does not satisfy "reach a state such that the proposition a is true" starting from $w[i:]$, the i-th output of lim[`eventually`] is `False`. Therefore, if $n_i^* = i$, the i-th output matches.

When the input satisfies "reach a state such that the proposition a is true" starting from $w[i:]$, i-th output of `eventually` is also `True` must also be `True`. if n_i^* equals the hitting time of the target start starting from $w[i:]$. In other words, n_i^* is upper bounded by $i + T_{\text{hit}}(w[i:])$.

Combing the two cases, I have $n_i^* \leq i + T_{\text{hit}}^0(w[i:])$, where $T_{\text{hit}}^0(w[i:])$ is the hitting time of the target state from $w[i:]$ if it is finite (i.e., the target state is reachable from $w[i:]$), or 0 otherwise. Using this bound on n_i^*, I derive the following chain of inequalities:

$$\begin{aligned}
\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)] &\leq \mathsf{E}[\max_{i \in \{1 \dots \mathsf{n}_1(n)\}} n_i^*] \\
&\leq \mathsf{E}[\max_{i \in \{1 \dots \mathsf{n}_1(n)\}} i + T_{\text{hit}}^0(w[i:])] \\
&\leq \mathsf{n}_1(n) + \mathsf{E}[\max_{i \in \{1 \dots \mathsf{n}_1(n)\}} T_{\text{hit}}^0(w[i:])] \\
&\leq \mathsf{n}_1(n) + \sum_{i=1}^{\mathsf{n}_1(n)} \mathsf{E}[T_{\text{hit}}^0(w[i:])]
\end{aligned}$$

The term $\mathsf{E}[T_{\text{hit}}^0(w[i:])]$ is the expected hitting time of the target state from the state $w[i:]$. However, it is upper bounded by $M = \max_{s \in S} T_{\text{hit}}^0(s)$, where $T_{\text{hit}}^0(s)$ is the hitting time of the target state from the state s if it is finite, or 0 otherwise. The quantity M is a constant given the Markov chain. Therefore, the numerator is further upper bounded by

$$\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)] \leq (1 + M)\, \mathsf{n}_1(n)$$

Plugging the above bound back to ratio in Equation (6.4): If $\mathsf{n}_2(n)$ grows faster than $\mathsf{n}_1(n)$ then the ratio approaches zero:

$$\lim_{n \to \infty} \frac{\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)]}{\mathsf{n}_2(n)} \leq (1 + M) \lim_{n \to \infty} \frac{\mathsf{n}_1(n)}{\mathsf{n}_2(n)} = 0$$

and the objective is limit PAC-learnable. □

6.5.5 Proof of Theorem 6.4.2

Proof. By the sufficient condition for limit PAC-learnability (Section 6.3), I need to show that the following condition holds for all $\epsilon > 0$:

$$\lim_{n\to\infty} \mathsf{P}(|\texttt{always0}(\mathsf{n}_1(n), w), \lim[\texttt{always0}](w)| > \epsilon) = 0, \text{ and}$$

$$\lim_{n\to\infty} \mathsf{P}(|\texttt{reach\_and\_retraces}(\mathsf{n}_2(n), w), \lim[\texttt{reach\_and\_retraces}](w)| > \delta^*_{\texttt{always0}}(\mathsf{n}_1(n), \epsilon)) = 0,$$

where the the two conditions are due to $i = 1$ and $i = 2$ in the theorem statement, respectively. The first condition is satisfied by letting $\mathsf{n}_1(n)$ grow to infinity:

$$\lim_{n\to\infty} \mathsf{n}_1(n) = \infty$$

so I focus on the second condition.

The function $\delta^*_{\texttt{always0}}$ is the modulus of continuity of the function `always0`. Since the function `always0` inspects only the first n elements of the input stream, and the input stream is the metric space over the infinite-length words $\mathbb{B}^\omega$ which has the metric $d_{\mathbb{B}^\omega}(w_1, w_2) = 2^{-L_{\text{prefix}}(w_1,w_2)}$, where $L_{\text{prefix}}(w_1, w_2)$ is the length of the longest common prefix of w_1 and w_2, it is simply $\delta^*_{\texttt{always0}}(n, \epsilon) = 2^{-n}$. Thus, the condition simplifies to

$$\lim_{n\to\infty} \mathsf{P}(|\texttt{reach\_and\_retraces}(\mathsf{n}_2(n), w), \lim[\texttt{reach\_and\_retraces}](w)| > 2^{-\mathsf{n}_1(n)}) = 0.$$

Recall the following epsilon-delta definition of the limit of the function `reach_and_retraces`:

$$\lim[\texttt{reach\_and\_retraces}](w) = \lim_{n\to\infty} \texttt{reach\_and\_retraces}(n, w)$$

For all $\epsilon > 0$ and $w \in \mathbb{B}^\omega$, there exists n such that for all $n' \geq n$:

$$2^{-L_{\text{prefix}}(\texttt{reach\_and\_retraces}(n',w), \lim[\texttt{reach\_and\_retraces}](w))} < \epsilon$$

Let $n^*(\epsilon, w)$ be the minimum n that satisfies the above for any given $w \in \mathbb{B}^\omega$, called the *local modulus of convergence* of `reach_and_retraces` to $\lim[\texttt{reach\_and\_retraces}]$. Then I rewrite the above inequaility using the local modulus of convergence:

$$\begin{aligned}
&\mathsf{P}(|\texttt{reach_and_retraces}(\mathsf{n}_2(n), w), \lim[\texttt{reach_and_retraces}](w)| > 2^{-\mathsf{n}_1(n)}) \\
&\leq \mathsf{P}(n^*(2^{-\mathsf{n}_1(n)}, w) \geq \mathsf{n}_2(n)) \\
&\leq \frac{\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)]}{\mathsf{n}_2(n)}
\end{aligned}$$

The second step is due to Markov's inequality.

Therefore, by Theorem 6.3.2, it suffices to show that the ratio approaches zero in the limit of n:

$$\lim_{n\to\infty} \frac{\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)]}{\mathsf{n}_2(n)} = 0 \tag{6.5}$$

Here, the numerator is the expected number of steps of the inputs w, so that the first n_1 elements of the output of `reach_and_retraces` matches that of lim[`reach_and_retraces`], where the expectation is taken over the distribution of the inputs w induced by the Markov chain. I show that the numerator is $\mathcal{O}(\mathsf{n}_1(n))$.

Consider the i-th output of the function `reach_and_retraces`, for $i \in \{1 \dots \mathsf{n}_1(n)\}$. Let n_i^* denote the minimum number of inputs `reach_and_retraces` consumes such that the i-th output matches. Two cases arise as follows.

When the input violates "reach the goal and retrace the path" starting from $w[i{:}]$, the i-th output of lim[`reach_and_retraces`] is `False`. Thus, if $n_i^* = i$, the i-th output matches.

When the input satisfies "reach the goal and retrace the path" starting from $w[i{:}]$, i-th output of `reach_and_retrace` is also `True` must also be `True`. if n_i^* equals twice the hitting time of the target start starting from $w[i{:}]$ (it is twice because once to reach the target state and once to retrace the path). In other words, n_i^* is upper bounded by $i + 2 \cdot T_{\text{hit}}(w[i{:}])$.

Combing the two cases, I have $n_i^* \leq i + 2 \cdot T^0_{\text{hit}}(w[i{:}])$, where $T^0_{\text{hit}}(w[i{:}])$ is the hitting time of the target state from $w[i{:}]$ if it is finite (i.e., the target state is reachable from $w[i{:}]$), or 0 otherwise. Using this bound on n_i^*, I derive the following chain of inequalities:

$$\begin{aligned}
\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)] &\leq \mathsf{E}[\max_{i\in\{1\dots\mathsf{n}_1(n)\}} n_i^*] \\
&\leq \mathsf{E}[\max_{i\in\{1\dots\mathsf{n}_1(n)\}} i + 2 \cdot T^0_{\text{hit}}(w[i{:}])] \\
&\leq \mathsf{n}_1(n) + 2 \cdot \mathsf{E}[\max_{i\in\{1\dots\mathsf{n}_1(n)\}} T^0_{\text{hit}}(w[i{:}])] \\
&\leq \mathsf{n}_1(n) + 2 \cdot \sum_{i=1}^{\mathsf{n}_1(n)} \mathsf{E}[T^0_{\text{hit}}(w[i{:}])]
\end{aligned}$$

The term $\mathsf{E}[T^0_{\text{hit}}(w[i{:}])]$ is the expected hitting time of the target state from the state $w[i{:}]$. However, it is upper bounded by $M = \max_{s\in S} T^0_{\text{hit}}(s)$, where $T^0_{\text{hit}}(s)$ is the hitting time of the target state from the state s if it is finite, or 0 otherwise. The quantity M is a constant given the Markov chain. Therefore, the numerator is further upper bounded by

$$\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)] \leq (1 + 2M)\, \mathsf{n}_1(n)$$

Plugging the above bound back to ratio in Equation (6.5): If $\mathsf{n}_2(n)$ grows faster than $\mathsf{n}_1(n)$

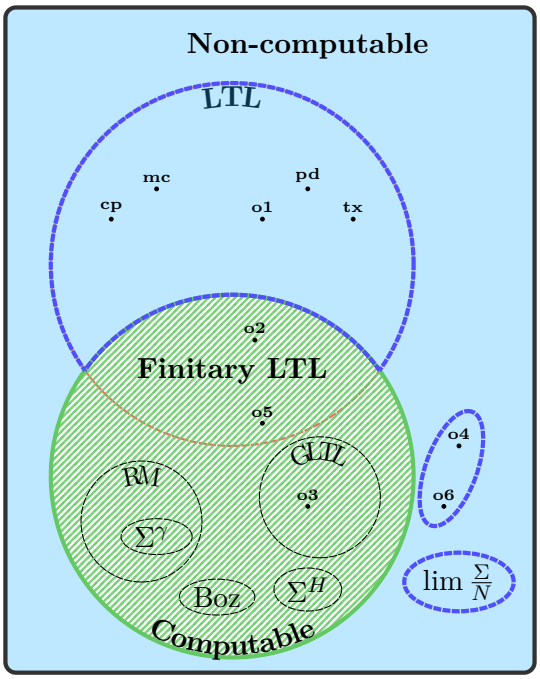


Figure 6.3: Landscape of objectives' learnability up to the current chapter. Dashed circles represent classes of objectives. The blue area denotes non-computable objectives. This chapter introduced limit PAC-learnability as a suitable notion of learnability guarantee, along with a sufficient condition that helps establish it. The example objectives in this region have provably limit PAC-learnable realizations. The green slanted area represents computable objectives that are PAC-learnable.

then the ratio approaches zero:

$$\lim_{n\to\infty} \frac{\mathsf{E}[n^*(2^{-\mathsf{n}_1(n)}, w)]}{\mathsf{n}_2(n)} \leq (1+2M) \lim_{n\to\infty} \frac{\mathsf{n}_1(n)}{\mathsf{n}_2(n)} = 0$$

and the objective is limit PAC-learnable. □

6.6 Chapter Summary

This chapter presents my results on non-computable reinforcement-learning objectives. I provide the first universal representation, a limit PAC-learnability criterion tailored for non-computable objectives, and a sufficient condition to establish their learnability. The results in this chapter lay the groundwork for learning policies with guarantees for non-computable reinforcement-learning objectives. The universal representation establishes a formal foundation for expressing any non-computable objective. The limit PAC-learnability criterion provides a learnability guarantee, while the sufficient condition offers a structured frame-

work for analyzing and proving the learnability of these objectives.

Figure 6.3 illustrates the landscape of objective learnability up to this chapter. The green slanted area highlights computable objectives that are PAC-learnable, as established in Chapter 5. Beyond this area lie non-computable objectives. This chapter introduces a universal representation of these objectives, a learnability criterion, and a sufficient condition for proving their learnability — depicted as the light blue area in the figure. Using this condition, I demonstrate that various example objectives and classes of objectives, enclosed by the dashed blue boundaries in the figure are limit PAC-learnable. Specifically, infinite-horizon LTL formulas, the limit average reward objective, and the example objectives **o4** (reach and retrace, Section 6.4.1.4), and **o6** (repeat reach and retrace, Section 6.4.2.2) all satisfy limit PAC-learnability.

Application to Other Objectives Although I demonstrated these results through examples, the methods apply broadly to both existing and new non-computable objectives. To establish the limit PAC-learnability of a non-computable objective, one first express it as a nested limit of a sequence of computable functions. If the expression involves only a single limit, the sufficient condition ensures limit PAC-learnability. For objectives involving multiple limits, one may decompose the objective into a composition of non-computable functions, each represented as a single limit of computable functions. The sufficient condition then guides the reasoning process by constraining the choice of indices for each function in the composition to ensure overall convergence. Identifying suitable indices that satisfy the condition proves the limit PAC-learnability of the objective through the constructed realization.

Generalization to Compact Policy Classes While this chapter assumes a fixed finite set of deterministic policies and the planning-with-generated-model setting, the core result extends to compact policy classes, such as stochastic finite-memory policies, and reinforcement learning settings, if PAC-learning algorithms exist for these settings. Future work could develop PAC-learning algorithms for such settings, and my results would then provide a foundation for learning non-computable objectives in these settings.

Chapter 7

Discussion

The research presented in this thesis addresses a fundamental gap in reinforcement learning: the specification and learnability of general objectives. By moving beyond reward-based paradigms, this thesis develops a theoretical foundation for specifying and learning objectives encompassing a broader spectrum of complexity, including non-computable objectives. These contributions lay the groundwork for advancing our understanding of reinforcement-learning objectives and extend the potential of intelligent agents under principled guarantees in a wide range of applications. This chapter explores the implications of this work in the context of AI safety and alignment, highlighting its relevance to contemporary challenges. Additionally, it outlines future research directions that build upon the findings presented in this thesis.

7.1 Advancing AI Safety and Alignment

Artificial intelligence (AI) research has experienced exponential growth in the past decade, with reinforcement learning at the forefront of this growth. As AI systems become increasingly capable, the importance of AI safety and alignment with human values has become a central concern. [57]. A body of work dedicated to improving reward-based approaches to alignment, such as inverse reinforcement learning [68, 69], reward shaping [70], and learning reward functions interactively from human feedback [71]. These efforts aim to improve AI systems' alignment with user intent by refining reward functions to more closely approximate desired outcomes. Indeed, approaches like RLHF have demonstrated remarkable success, particularly in large language models (LLMs), where they serve as a critical mechanism for aligning system behavior with human values.

However, reward-based approaches remain limited in formally defining objectives and guaranteeing alignment. This thesis extends beyond rewards functions and encompasses the broader class of general objectives. The foundational contributions presented in this thesis

enable the specification of formally defined objectives that enhance alignment with human values and offer guarantees for learning policies that effectively maximize these objectives.

The underlying premise of this thesis is that designing a truly safe and aligned AI system requires two fundamental capabilities: first, the ability to specify objectives that reflect the user's intent formally, and second, the ability to learn policies that maximize these objectives with guarantees. While these requirements might be overly stringent or unnecessary in some applications — such as "generating text that is humorous" where the objectives are challenging to specify formally, or scenarios when one does not need a near-optimal performance but merely one that is good enough (e.g., performing) better than the average human – they are essential in high-stake domains. This extension is particularly critical in high-stakes domains such as autonomous driving, healthcare, and finance, where a high level of assurance of AI safety and alignment with intent. By laying the groundwork for specifying and optimizing general objectives, this thesis contributes directly to the broader mission of advancing AI safety and alignment in an era of rapidly evolving capabilities.

7.2 Future Work

I outline problems that I did not address in this thesis; nonetheless, they might be useful directions for future research.

7.2.1 Algorithm Efficiency

The focus of this thesis has primarily been on the computability of the objective, with less emphasis on algorithmic efficiency with respect to the objective. Specifically, the assumption that the given objective is a constant implies that the measures of the learning algorithm's efficiency, such as sample complexity and computational complexity, do not vary with the objective. In practice, the efficiency of learning algorithms with respect to the objective is an important consideration, as it determines the practicality of the approach for specific classes of objectives. Future research should address how to provide guarantees on the efficiency of learning algorithms with respect to the objectives, for example, by providing sample complexity bounds that depend on the complexity of the objective. A key challenge lies in the variability introduced by different representation languages for objectives, as the same objective may exhibit different expressiveness in distinct languages. Consequently, fixing the objectives language is necessary to establish consistent measures of complexity.

7.2.2 Identifying Good Policies in a Compact Policy Class

Chapter 6 of thesis assumes the availability of an algorithm for a computable objective that identifies a near-optimal policy for the objective in a compact set of policies. While such an algorithm exists for planning-with-a-generative-model setting and a finite set of deterministic policies, algorithms for other settings remain unvisited. Since the majority of reinforcement learning research had focused on reward-based objectives, and reward-based objectives have the property that there always exists a deterministic stationary optimal policy (for discounted rewards) or a deterministic finite-memory optimal policy (for finite-horizon rewards), the question of identifying good policies in an arbitrary policy class has been less explored in the literature. The results in Chapter 6 suggest future exploration into expression of a compact set of policies and algorithms that identify good policies in these compact policy classes, thereby extending the applicability of my results to broader settings.

7.2.3 Towards an Objectives Programming Systsem

The theoretical results presented in this thesis provide insights into the design of a programming language for expressing general reinforcement-learning objectives. For computable objectives, a potential language could offer constructs for expressing computable functions over real numbers. For non-computable objectives, it could include constructs for specifying nested limits of computable functions, similar to the universal representation introduced in this chapter. Such a language would provide a principled way to specify objectives that are provably learnable by reinforcement learning algorithms. Moreover, akin to compilers that translate high-level programming languages into machine code, the language could compile to learning algorithms that identify good policies that achieve the specified objectives with guarantees. The design of such a language is challenging, requiring a balance between expressiveness and learnability. The results in this thesis provide a theoretical foundation for this task.

7.3 Conclusion

Artificial intelligence systems are growing in capabilities and deployed in various applications. Some applications require high assurance that the AI system will perform as intended in the presence of uncertainty in the environment. The majority of reinforcement learning research has focused on reward-based objectives. As a result, the standard practice has focused on designing reward functions as surrogates for the true objectives — specified informally to align with the user's intent — and then learning a policy using a reinforcement-

learning algorithm. However, this approach lacks a formal specification of the true objective and offers little assurance that the resulting policy will perform well with respect to the true objective. Despite these limitations, reward-based methods have achieved remarkable practical success in domains such as robotics, games, and natural language processing. Yet, this raises a critical question: Are we willing to accept the risks associated with the lack of formal guarantees, especially in high-stakes applications?

This thesis takes a different approach by moving beyond rewards and considering the general class of objectives, and acts as a first step toward addressing this fundamental gap in reinforcement learning. The results presented in this thesis provide a theoretical foundation for specifying and learning objectives that encompass a broader spectrum of complexity, including non-computable objectives. Ideally, objectives should be specified formally, and efforts should focus on reducing them to objectives that are efficiently learnable by reinforcement learning algorithms. This approach ensures both formal guarantees of the objective and the efficiency of the learning algorithm. While this thesis does not provide an end-to-end solution to this challenge, it lays the theoretical groundwork for a deeper understanding of reinforcement-learning objectives. It presents a step toward expanding the capabilities of intelligent agents under principled guarantees.

Even if the realization of fully automated systems remains far in the future, the results in this thesis still provide insights for practitioners interested in designing reward functions. Practitioners should first consider if their objective, even if informally specified, is computable — that is, whether it is possible to constructively assign a real number to each trajectory of the system. If the objective is computable, the findings in Chapter 5 guide the design of reward functions that precisely capture the objective. For objectives that are not computable, practitioners can consider expressing the objective as a nested limit of a sequence of computable functions. In such cases, they can then design a reward function that aligns with the computable functions and use learning algorithms that learn good policies for each computable function in the sequence, with the expectation that the policy will perform well concerning the true objective in the limit.

With the rise of AI systems, such as large language models, being deployed to millions of users — each with their objectives and preferences — this work is particularly timely. These systems are increasingly relevant in high-stakes applications such as healthcare, finance, and autonomous driving, where the ability to formally specify and learn objectives is critical. The results presented in this thesis will inspire future research in this direction, and the theoretical foundation laid here will serve as a stepping stone toward a deeper understanding of the specification of formal objectives in reinforcement learning.

Appendix A

Pseudocode and Utility Functions

A.1 Standard Data Types Used in Psuedocode

In the pseudocode, I use the following standard data types:

- `Union[A, B]` A union of types, either type A or type B,
- `List[A]` A list of elements of type A, representing a finite-length word,
- `Iterator[A]` An iterator over elements of type A, representing an infinite-length word. Note that to avoid confusion, `Iterator[A]` always represents an infinite-length word, although Python's `Iterator` type can represent finite-length words as well,
- `Tuple[A, B]` A tuple of elements of types A and B, representing a pair of elements,
- `Callable[[A], B]` A function that takes an argument of type A and returns a value of type B.

A.2 Programming Utilities

Listing A.1 presents standard functions over finite objects used in the thesis. All these functions are implementable in any Turing-complete programming language.

Listing A.2 presents standard stream and sequence processing utilities used in the thesis. These functions are standard in Python's standard library [72], and uses Python syntax for generator, iterator and stream processing. Nonetheless, these functions are implementable in any Turing-complete programming language.

```
def any(w: List[Boolean]) -> Boolean:
  """Return True if any element of the input list is True."""

def all(w: List[Boolean]) -> Boolean:
  """Return True if all elements of the input list are True."""

def max(w: List[Rational]) -> Rational:
  """Return the maximum element of the input list."""

def min(w: List[Rational]) -> Rational:
  """Return the minimum element of the input list."""

def log2ceil(x: Rational) -> Integer:
  """Return the smallest integer greater than or equal to the base-2 logarithm of x."""

def log2floor(x: Rational) -> Integer:
  """Return the largest integer less than or equal to the base-2 logarithm of x."""
```

Listing A.1: Standard function over finite objects. Standard arithmetic operations on naturals, integers and rationals omitted.

```
def next(w: Iterator[A]) -> A:
  """Read the next element of the input stream, advancing the iterator.
  If the stream is empty, raise a StopIteration exception.
  """

def map(f: Callable[[A], B], w: Iterator[A]) -> Iterator[B]:
  """Apply the function f to each element of the input stream."""

def enumerate(w: Iterator[A]) -> Iterator[Tuple[Natural, A]]:
  """Enumerate the input stream with indices."""

def zip(*wi: Iterator[Ti]) -> Iterator[Tuple[Ti, ...]]:
  """Transpose two or more input streams into a single stream of tuples."""

def islice(w: Iterator[A], n: Natural) -> Iterator[A]:
  """Slice the first n elements of the iterator.
     The returned iterator has the property that trying to read
     more than n elements raises a StopIteration exception.
  """

def tee(w: Iterator[A], n: Natural = 2) -> Tuple[Iterator[A], ...]:
  """Create n independent copies (default two) of the input iterator."""

def count(n: Integer = 0) -> Iterator[Integer]:
  """Generate an infinite stream of integers starting from n (default 0):
         n, n+1, n+2, ...
  """

def repeat(x: A) -> Iterator[A]:
  """Generate a stream of x repeated infinitely many times:
         x, x, x, ...
  """
```

Listing A.2: Standard stream and sequence processing utilities. The functions match closely with the Python standard library, and they are implementable in any Turing-complete programming language.

Appendix B

Additional Empirical Justification to Section 4.4

This section completes the empirical justification Section 4.4 with additional details on the empirical experiments.

Previous work has introduced various reinforcement-learning algorithms for LTL objectives [1, 2, 16, 17]. I therefore ask the research question: *Do the sample complexities for reinforcement-learning algorithms for LTL objectives introduced by previous work depend on the transition probabilities of the environment?*

To answer the above question, I consider a set of reinforcement-learning algorithms for LTL objectives and empirically measure the sample size for each algorithm to obtain a near-optimal policy with high probability.

B.1 Methodology

Reinforcement-learning algorithms We consider a set of recent reinforcement-learning algorithms for LTL objectives [1, 16], about which I give more details in Appendix B.4. These algorithms are all implemented in the Mungojerrie toolbox [73].

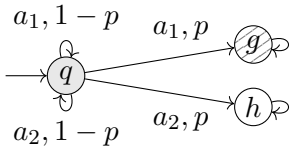


Figure B.1: One of the two environment MDPs used in the experiments.

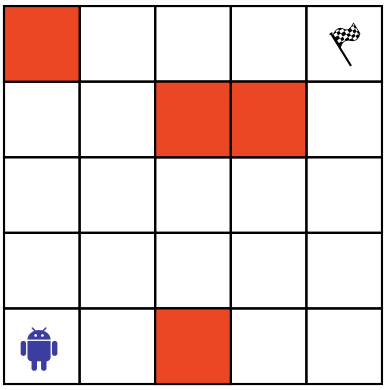


Figure B.2: Gridworld environment MDP from Sadigh et al. [2] with a customized transition dynamics. The agent starts from the lower left corner. At each time step, the agent can choose to move up, down, left or right. The white cells are sticky: the agent moves towards the intended direction with probability $1-p$ (or stays stationary if it will move off the grid), and stays stationary with probability p. The red cells are trapping: once the agent steps on a red cell, it stays there forever.

Objectives and Environment MDPs We consider two pairs of LTL formulas and environment MDPs (LTL-MDP pair). The first pair is the formula $\mathsf{F}\, h$ and the counterexample MDP constructed according to Section 4.3.3.1, shown in Figure B.1. The second pair is the formula $\mathsf{F}\, \mathit{goal}$ and a gridworld environment MDP from the case study by Sadigh et al. [2] with a customized transition dynamics, shown in Figure B.2.

Experiment Methodology We ran the considered algorithms on each chosen LTL-MDP pair with a range of values for the parameter p and let the algorithms perform N environment samples.

For each algorithm and each pair of values of p and N, I fix $\epsilon = 0.1$ and repeatedly run the algorithm to obtain a Monte Carlo estimation of the LTL-PAC probability (left side of Equation (4.2)) for that setting of p, N and ϵ. We repeat each setting until the estimated standard deviation of the estimated probability is within 0.01. In the end, for each algorithm and LTL-MDP pair I obtain $5 \times 21 = 105$ LTL-PAC probabilities and their estimated standard deviations.

For the first LTL-MDP pair, I vary p by a geometric progression from 10^{-1} to 10^{-3} in 5 steps: $p(i) = 10^{-\frac{i+1}{2}}$ for $1 \le i \le 5$. We vary N by a geometric progression from 10^1 to 10^5 in 21 steps.

For the second LTL-MDP pair, I vary p by a geometric progression from 0.9 to 0.6 in 5 steps: $p(i) = 0.9 \times 0.903^{-i}$ for $1 \le i \le 5$. We vary N by a geometric progression from 3540 to 9×10^4 in 21 steps; if an algorithm does not converge to the desired LTL-PAC probability within 9×10^4 steps, I rerun the experiment with an extended range of N from 3540 to

Reinforcement-learning-algorithm	Learning Rate	Exploration	Reset Episode Every Steps
Q-learning	$\frac{10}{10+t}$	Linear decay from 1.0 to 10^{-1}	10
Double Q-learning	$\frac{30}{30+t}$	Linear decay from 1.0 to 10^{-1}	10
SARSA(λ)	$\frac{10}{10+t}$	Linear decay from 1.0 to 10^{-3}	10

Table B.1: Non-default hyper-parameters used for each learning-algorithm

1.5×10^5.

B.2 Results

Figure 4.4 presents the results for the algorithm in Bozkurt et al. [1] with the setting of Multi-discount, Q-learning, and the first LTL-MDP pair.

On the left, I plot the LTL-PAC probabilities vs. the number of samples N, one curve for each p. On the right, I plot the intersections of the curves in the left plot with a horizontal cutoff of 0.9.

As I see from the left plot of Figure 4.4, for each p, the curve starts at 0 and grows to 1 in a sigmoidal shape as the number of samples increases. However, as p decreases, the MDP becomes harder: As shown on the right plot of Figure 4.4, the number of samples required to reach the particular LTL-PAC probability of 0.9 grows exponentially.

Figure B.3 presents the complete results for all settings for the first LTL-MDP pair, and Figure B.4 present the complete results for all settings for the second LTL-MDP pair. These results are similar and lead to the same analysis as above.

B.3 Result Interpretation

Since the transition probabilities (p in this case) are unknown in practice, one can't know which curve in the left plot a given environment will follow. Therefore, given any finite number of samples, these reinforcement-algorithms cannot provide guarantees on the LTL-PAC probability of the learned policy. This result supports Theorem 4.3.4.

B.4 Empirical Experiment Details

Chosen Algorithms We consider a set of recent reinforcement-learning algorithms for LTL objectives implemented in the Mongujerrie toolbox [73].

A common pattern in these previous works [1, 2, 16] is that each work constructs a product MDP with rewards (i.e., an MDP with a reward function on that MDP) from

an LTL formula and an environment MDP. Moreover, these works permit the use of any standard reinforcement-learning algorithm, such as Q-learning or SARSA(λ), to solve the constructed product MDP with the specified reward function to obtain the product MDP's optimal policy. Finally, these works cast the optimal policy back to a non-Markovian policy of the environment MDP, which becomes the algorithm's output policy.

Following Hahn et al. [73], I call each specific construction of a product MDP with rewards as a *reward-scheme*. We then characterize each reinforcement-learning algorithm as a "reward-scheme" and "learning-algorithm" pair. We consider a total of five reward-schemes [1]: Reward-on-acc [2], Multi-discount [1], Zeta-reach [16], Zeta-acc [53], and Zeta-discount [53]. We consider a total of three learning-algorithms: Q-learning [74], Double Q-learning [75], and SARSA(λ) [76]. This yields a total of 15 reinforcement-learning algorithms for LTL objectives.

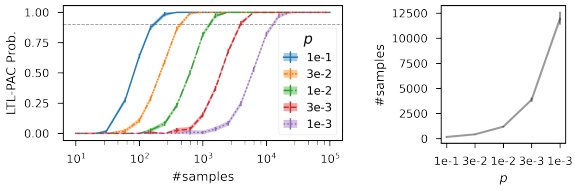
Algorithm Parameters Each reinforcement-learning algorithm in Mungojerrie accepts a set of hyper-parameters. For the majority of the hyper-parameters, I use their default values as in Mungojerrie Version 1.0 [73]. We present the hyper-parameters that differ from the default values in Table B.1. For each of the hyper-parameters in Table B.1, I use a different value from the default value because it allow all the algorithms that I consider to converge within 10^5 steps (i.e., the maximum learning steps that I allow). For SARSA(λ), I use $\lambda = 0$.

Software and Platform We use a custom version of Mungojerrie. My modifications are:

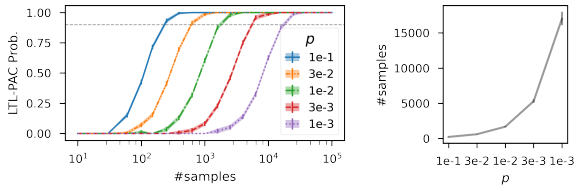
- Modification to allow parallel Monte Carlo estimation of the LTL-PAC probability.
- Modification to allow the reinforcement-learning algorithms to have a non-linear learning rate decay. In particular, I use a learning rate of $\frac{k}{k+t}$ at every learning step t, where k is a hyper-parameter (see Table B.1 for the value of k for each algorithm). This modification is necessary for ensuring Q-learning's convergence [74].

I run all experiments on a machine with 2.9 GHz 6-Core CPU and 32 GB of RAM.

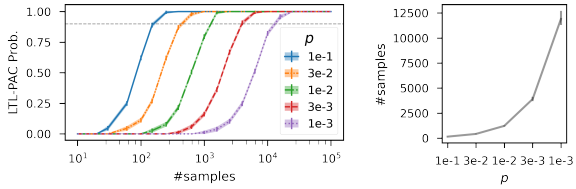
[1] We use the same naming of each reward-scheme as in the Mungojerrie toolbox [73]



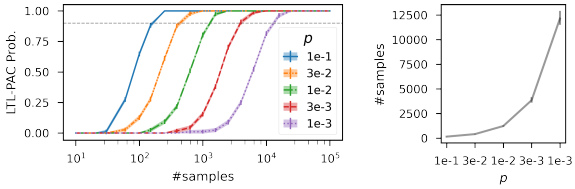
(a) Reward-on-acc with Q-learning



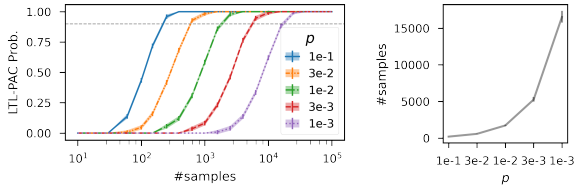
(b) Reward-on-acc with Double Q-learning



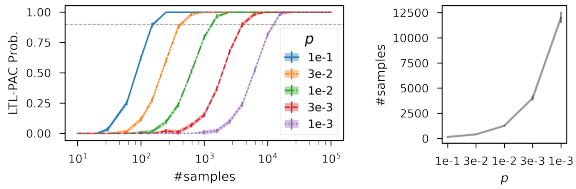
(c) Reward-on-acc with SARSA(λ)



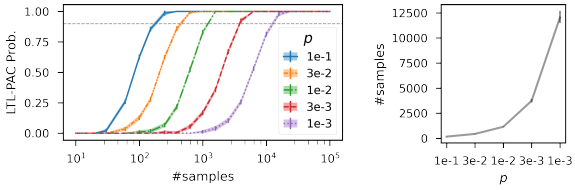
(d) Multi-discount with Q-learning



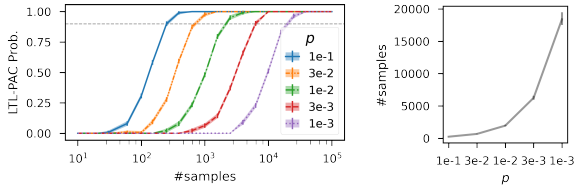
(e) Multi-discount with Double Q-learning



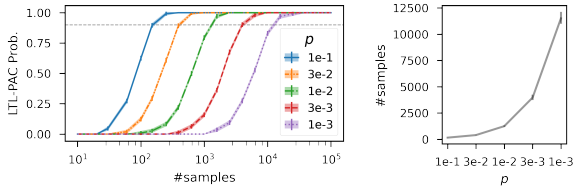
(f) Multi-discount with SARSA(λ)



(g) Zeta-reach with Q-learning

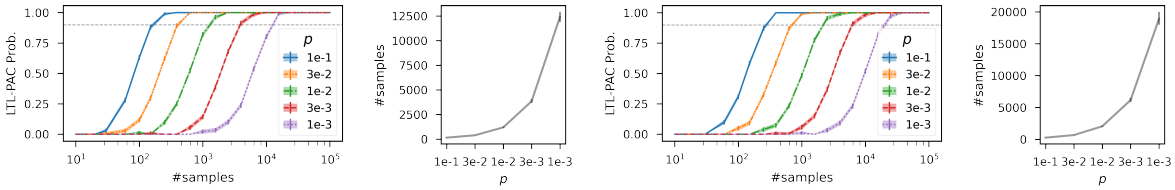


(h) Zeta-reach with Double Q-learning



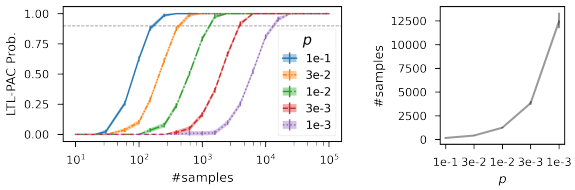
(i) Zeta-reach with SARSA(λ)

Figure B.3: Empirical results of the first LTL-MDP pair (continued on next page)

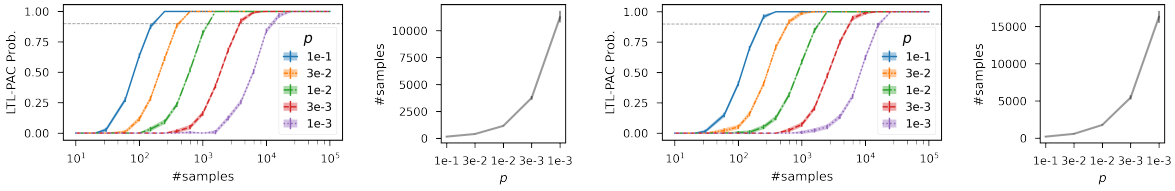


(j) Zeta-acc with Q-learning

(k) Zeta-acc with Double Q-learning

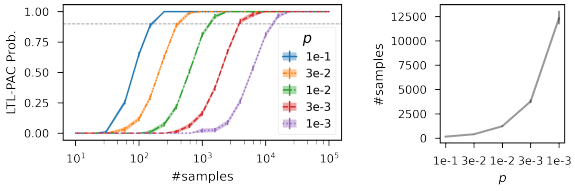


(l) Zeta-acc with SARSA(λ)



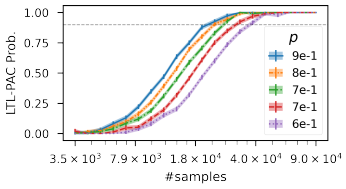
(m) Zeta-discount with Q-learning

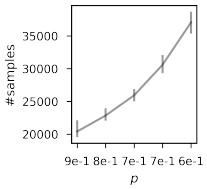
(n) Zeta-discount with Double Q-learning



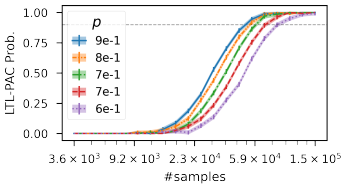
(o) Zeta-discount with SARSA(λ)

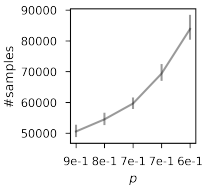
Figure B.3: Empirical results of the first LTL-MDP pair (continued). Each sub-figure corresponds to a specific reward-scheme and learning-algorithm pair. For each sub-figure, on the left: LTL-PAC probabilities vs. number of samples, for varying parameters p; on the right: number of samples needed to reach 0.9 LTL-PAC probability vs. parameters p.



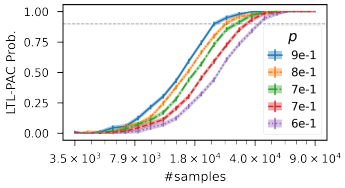


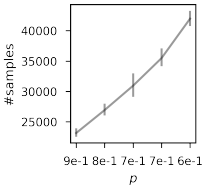
(a) Reward-on-acc with Q-learning



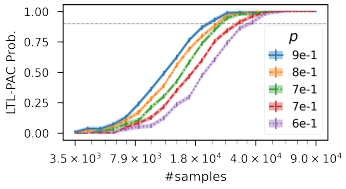


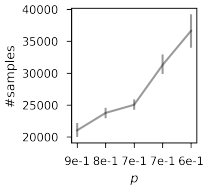
(b) Reward-on-acc with Double Q-learning



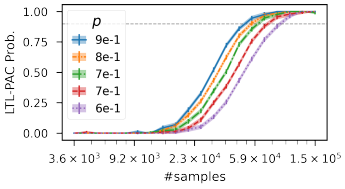


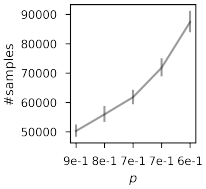
(c) Reward-on-acc with SARSA(λ)



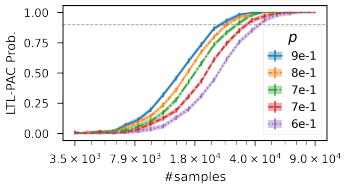


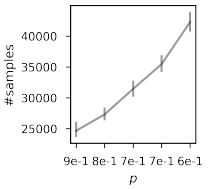
(d) Multi-discount with Q-learning



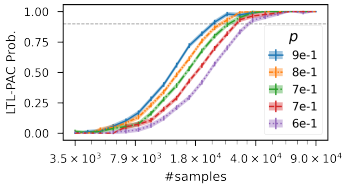


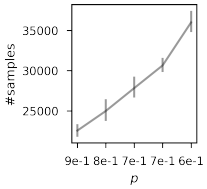
(e) Multi-discount with Double Q-learning



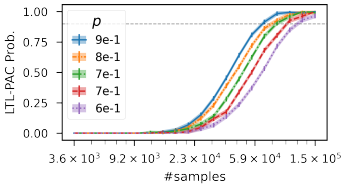


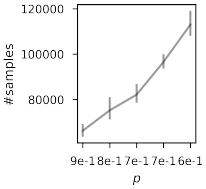
(f) Multi-discount with SARSA(λ)



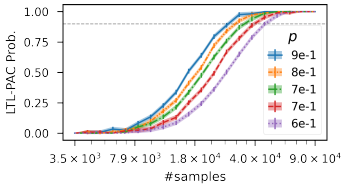


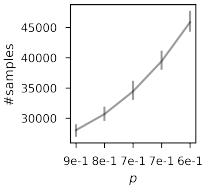
(g) Zeta-reach with Q-learning





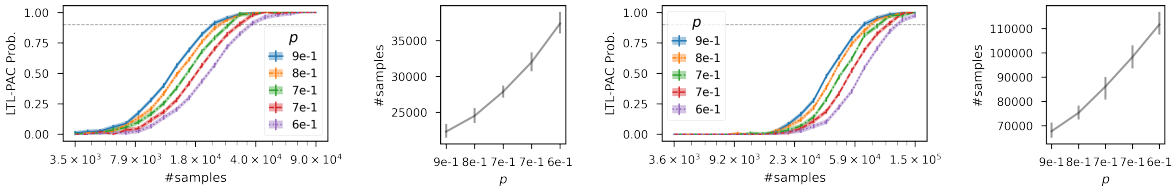
(h) Zeta-reach with Double Q-learning





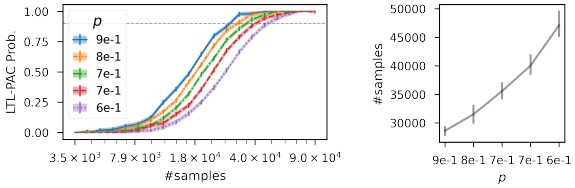
(i) Zeta-reach with SARSA(λ)

Figure B.4: Empirical results of the second LTL-MDP pair (continued on next page)

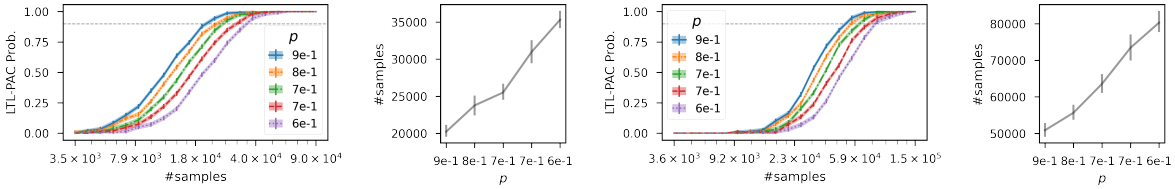


(j) Zeta-acc with Q-learning

(k) Zeta-acc with Double Q-learning

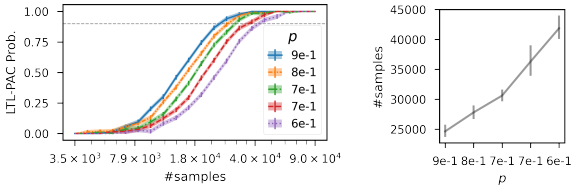


(l) Zeta-acc with SARSA(λ)



(m) Zeta-discount with Q-learning

(n) Zeta-discount with Double Q-learning



(o) Zeta-discount with SARSA(λ)

Figure B.4: Empirical results of the second LTL-MDP pair (continued). Each sub-figure corresponds to a specific reward-scheme and learning-algorithm pair. For each sub-figure, on the left: LTL-PAC probabilities vs. number of samples, for varying parameters p; on the right: number of samples needed to reach 0.9 LTL-PAC probability vs. parameters p.

References

[1] A. Bozkurt, Y. Wang, M. Zavlanos, and M. Pajic. "Control Synthesis from Linear Temporal Logic Specifications using Model-Free Reinforcement Learning". In: *International Conference on Robotics and Automation*. 2020.

[2] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia. "A Learning Based Approach to Control Synthesis of Markov Decision Processes for Linear Temporal Logic Specifications". In: *Conference on Decision and Control*. 2014.

[3] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN: 9780134610993. URL: http://aima.cs.berkeley.edu/.

[4] L. Ouyang et al. "Training language models to follow instructions with human feedback". In: *Neural Information Processing Systems*. Curran Associates Inc., 2022.

[5] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* (2016).

[6] J. M. Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* (2021).

[7] C. Yu, J. Liu, S. Nemati, and G. Yin. "Reinforcement Learning in Healthcare: A Survey". In: *ACM Computing Surveys* (2021).

[8] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. "Learning scheduling algorithms for data processing clusters". In: *Proceedings of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, 2019.

[9] O. M. Andrychowicz et al. "Learning dexterous in-hand manipulation". In: *International Journal of Robotics Research* (2020).

[10] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[11] A. Strehl, L. Li, E. Wiewiora, J. Langford, and M. Littman. "PAC Model-Free Reinforcement Learning". In: *International Conference on Machine Learning*. 2006.

[12] R. I. Brafman and M. Tennenholtz. “R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning”. In: *Journal of Machine Learning Research* 3 (2002).

[13] A. Pnueli. “The Temporal Logic of Programs”. In: *Symposium on Foundations of Computer Science*. 1977.

[14] J. Fu and U. Topcu. “Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints”. In: *Robotics: Science and Systems X*. 2014.

[15] X. Li, C. Vasile, and C. Belta. “Reinforcement learning with temporal logic rewards”. In: *International Conference on Intelligent Robots and Systems* (2017).

[16] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak. “Omega-Regular Objectives in Model-Free Reinforcement Learning”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. 2019.

[17] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. Pappas, and I. Lee. “Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees”. In: *Conference on Decision and Control*. 2019.

[18] D. Henriques, J. G. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. “Statistical Model Checking for Markov Decision Processes”. In: *International Conference on Quantitative Evaluation of Systems*. 2012.

[19] P. Ashok, J. Křetínský, and M. Weininger. “PAC Statistical Model Checking for Markov Decision Processes and Stochastic Games”. In: *Computer Aided Verification*. 2019.

[20] Y. Jiang, S. Bharadwaj, B. Wu, R. Shah, U. Topcu, and P. Stone. “Temporal-Logic-Based Reward Shaping for Continuing Learning Tasks”. In: *arXiv preprint: 2007.01498* (2020).

[21] M. L. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan. “Environment-Independent Task Specifications via GLTL”. In: *arXiv preprint: 1704.04341* (2017).

[22] C. Yang, M. Littman, and M. Carbin. “On the (In)Tractability of LTL Objectives for Reinforcement Learning”. In: *The International Joint Conference on Artificial Intelligence*. 2022.

[23] A. Camacho, R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. “LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning”. In: *The International Joint Conference on Artificial Intelligence*. 2019.

[24] G. D. Giacomo, L. Iocchi, M. Favorito, and F. Patrizi. "Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications". In: *International Conference on Automated Planning and Scheduling*. 2019.

[25] K. Jothimurugan, R. Alur, and O. Bastani. "A Composable Specification Language for Reinforcement Learning Tasks". In: *Neural Information Processing Systems*. 2019.

[26] A. Ronca and G. De Giacomo. "Efficient PAC Reinforcement Learning in Regular Decision Processes". In: *The International Joint Conference on Artificial Intelligence*. 2021.

[27] R. Alur, S. Bansal, O. Bastani, and K. Jothimurugan. "A Framework for Transforming Specifications in Reinforcement Learning". In: *arXiv preprint: 2111.00272* (2021).

[28] H. Bazille, B. Genest, C. Jegourel, and J. Sun. "Global PAC Bounds for Learning Discrete Time Markov Chains". In: *Computer Aided Verification*. 2020.

[29] C. Yang, M. Littman, and M. Carbin. "Computably Continuous Reinforcement-Learning Objectives are PAC-learnable". In: *National Conference on Artificial Intelligence*. 2023.

[30] G. De Giacomo and M. Y. Vardi. "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces". In: *The International Joint Conference on Artificial Intelligence*. 2013.

[31] C.-N. Fiechter. "Efficient Reinforcement Learning". In: *Conference on Computational Learning Theory*. 1994.

[32] M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[33] M. Kearns and S. Singh. "Near-Optimal Reinforcement Learning in Polynomial Time". In: *Machine Learning* 49.2 (2002).

[34] S. M. Kakade. "On the Sample Complexity of Reinforcement Learning". PhD thesis. Gatsby Computational Neuroscience Unit, University College London, 2003.

[35] T. Brázdil, K. Chatterjee, M. Chmelík, V. Forejt, J. Křetínský, M. Kwiatkowska, D. Parker, and M. Ujma. "Verification of Markov Decision Processes Using Learning Algorithms". In: *Automated Technology for Verification and Analysis*. 2014.

[36] L. G. Valiant. "A Theory of the Learnable". In: *Communications of the ACM* 27.11 (1984).

[37] C. Dann, T. Lattimore, and E. Brunskill. "Unifying PAC and regret: uniform PAC bounds for episodic reinforcement learning". In: *Neural Information Processing Systems*. 2017.

[38] C. Dann, L. Li, W. Wei, and E. Brunskill. "Policy Certificates: Towards Accountable Reinforcement Learning". In: *International Conference on Machine Learning*. 2019.

[39] M. Kearns, Y. Mansour, and A. Y. Ng. "Approximate Planning in Large POMDPs via Reusable Trajectories". In: *Neural Information Processing Systems*. 1999.

[40] J.-B. Grill, M. Valko, and R. Munos. "Blazing the trails before beating the path: Sample-efficient Monte-Carlo planning". In: *Neural Information Processing Systems*. 2016.

[41] A. L. Strehl, L. Li, and M. L. Littman. "Reinforcement Learning in Finite MDPs: PAC Analysis". In: *Journal of Machine Learning Research* 10 (2009).

[42] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[43] Z. Manna and A. Pnueli. "A Hierarchy of Temporal Properties". In: *Symposium on Principles of Distributed Computing*. 1987.

[44] T. Latvala. "Efficient Model Checking of Safety Properties". In: *Model Checking Software*. 2003.

[45] O. Kupferman and M. Vardi. "Model Checking of Safety Properties". In: *Formal Methods in System Design* 19 (1999).

[46] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. "Spot 2.0 - A Framework for LTL and ω -Automata Manipulation". In: *ATVA*. 2016.

[47] S. Temizer, M. Kochenderfer, L. Kaelbling, T. Lozano-Perez, and J. Kuchar. "Collision Avoidance for Unmanned Aircraft using Markov Decision Processes". In: *AIAA Guidance, Navigation, and Control Conference*. 2010.

[48] J. Kober, J. Bagnell, and J. Peters. "Reinforcement Learning in Robotics: A Survey". In: *The International Journal of Robotics Research* 32 (2013).

[49] W. Schwarting, J. Alonso-Mora, and D. Rus. "Planning and Decision-Making for Autonomous Vehicles". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018).

[50] S. Safra. "On the Complexity of ω-Automata". In: *Symposium on Foundations of Computer Science*. 1988.

[51] K. Weihrauch. *Computable Analysis: An Introduction*. Springer Science & Business Media, 2000.

[52] S. Sickert, J. Esparza, S. Jaax, and J. Křetínský. “Limit-Deterministic Büchi Automata for Linear Temporal Logic”. In: *Computer Aided Verification*. 2016.

[53] E. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak. “Faithful and Effective Reward Schemes for Model-Free Reinforcement Learning of Omega-Regular Objectives”. In: *Automated Technology for Verification and Analysis*. 2020.

[54] J. Corazza, I. Gavran, and D. Neider. “Reinforcement Learning with Stochastic Reward Machines”. In: *aaai*. 2022.

[55] R. Subramani, M. Williams, M. Heitmann, H. Holm, C. Griffin, and J. Skalse. “On the Expressivity of Objective-Specification Formalisms in Reinforcement Learning”. In: 2024.

[56] D. McDermott, M. Ghallab, A. E. Howe, C. A. Knoblock, A. Ram, M. M. Veloso, D. S. Weld, and D. E. Wilkins. *PDDL — the planning domain definition language*. 1998.

[57] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané. “Concrete Problems in AI Safety”. In: *arXiv preprint: 1606.06565* (2016).

[58] R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. “Teaching Multiple Tasks to an RL Agent using LTL”. In: *Adaptive Agents and Multi-Agent Systems*. 2018.

[59] Q. Gao, D. Hajinezhad, Y. Zhang, Y. Kantaros, and M. M. Zavlanos. “Reduced Variance Deep Reinforcement Learning with Temporal Logic Specifications”. In: *International Conference on Cyber-Physical Systems*. 2019.

[60] C. Voloshin, H. M. Le, S. Chaudhuri, and Y. Yue. “Policy optimization with linear temporal logic constraints”. In: *Neural Information Processing Systems*. 2024.

[61] J. Svoboda, S. Bansal, and K. Chatterjee. “Reinforcement Learning from Reachability Specifications: PAC Guarantees with Expected Conditional Distance”. In: *International Conference on Machine Learning*. 2024.

[62] A. Naik, R. Shariff, N. Yasui, H. Yao, and R. S. Sutton. “Discounted Reinforcement Learning Is Not an Optimization Problem”. In: *arXiv preprint: 1910.02140* (2019).

[63] D. Shao and M. Kwiatkowska. “Sample efficient model-free reinforcement learning from ltl specifications with optimality guarantees”. In: *The International Joint Conference on Artificial Intelligence*. 2023.

[64] X.-B. Le, D. Wagner, L. Witzman, A. Rabinovich, and L. Ong. “Reinforcement Learning with LTL and ω-Regular Objectives via Optimality-Preserving Translation to Average Rewards”. In: *Neural Information Processing Systems*. 2024.

[65] A. S. Kechris. *Classical Descriptive Set Theory*. Springer, 1995.

[66] M. Yasugi, T. Mori, and Y. Tsujii. “Effective properties of sets and functions in metric spaces with computability structure”. In: *Theoretical Computer Science* (1999).

[67] C. J. Bishop, E. A. Feinberg, and J. Zhang. “Examples concerning Abel and Cesàro limits”. In: *Journal of Mathematical Analysis and Applications* 2 (2014).

[68] A. Y. Ng and S. J. Russell. “Algorithms for Inverse Reinforcement Learning”. In: *International Conference on Machine Learning*. 2000.

[69] D. Hadfield-Menell, A. Dragan, P. Abbeel, and S. Russell. “Cooperative inverse reinforcement learning”. In: *Neural Information Processing Systems*. 2016.

[70] A. Y. Ng, D. Harada, and S. J. Russell. “Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping”. In: *International Conference on Machine Learning*. 1999.

[71] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. “Deep reinforcement learning from human preferences”. In: *Neural Information Processing Systems*. 2017.

[72] P. S. Foundation. *Python Documentation*. 2025. URL: https://docs.python.org/3/.

[73] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak. “Mungojerrie: Reinforcement Learning of Linear-Time Objectives”. In: *arXiv preprint: 2106.09161* (2021).

[74] C. J. C. H. Watkins and P. Dayan. “Q-learning”. In: *Machine Learning* 8.3 (1992).

[75] H. V. Hasselt. “Double Q-learning”. In: *Neural Information Processing Systems*. 2010.

[76] R. S. Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine Learning* 3.1 (1988).