Learning Probabilistic Temporal Logic Specifications for Stochastic Systems

Rajarshi Roy[1] , **Yash Pote**[2] , **David Parker**[1] and **Marta Kwiatkowska**[1]
[1]University of Oxford, Oxford OX1 2JD, UK
[2]National University of Singapore
rajarshi.roy@cs.ox.ac.uk,yashppote@gmail.com,{david.parker, marta.kwiatkowska}@cs.ox.ac.uk

Abstract

There has been substantial progress in the inference of formal behavioural specifications from sample trajectories, for example using Linear Temporal Logic (LTL). However, these techniques cannot handle specifications that correctly characterise systems with stochastic behaviour, which occur commonly in reinforcement learning and formal verification. We consider the passive learning problem of inferring a Boolean combination of probabilistic LTL (PLTL) formulas from a set of Markov chains, classified as either positive or negative. We propose a novel learning algorithm that infers concise PLTL specifications, leveraging grammar-based enumeration, search heuristics, probabilistic model checking and Boolean set-cover procedures. We demonstrate the effectiveness of our algorithm in two use cases: learning from policies induced by RL algorithms and learning from variants of a probabilistic model. In both cases, our method automatically and efficiently extracts PLTL specifications that succinctly characterize the temporal differences between the policies or model variants.

1 Introduction

Temporal logic is a powerful formalism used not only for writing correctness specifications in formal methods but also for defining non-Markovian goals and objectives in reinforcement learning (RL) and control tasks [Li *et al.*, 2017; Camacho *et al.*, 2019; Hasanbeig *et al.*, 2019; Bozkurt *et al.*, 2020]. Among temporal logics, Linear Temporal Logic (LTL) [Pnueli, 1977] is a de-facto standard for expressing temporal behaviours due to its widespread usage. The popularity of LTL stems from its desirable theoretical properties, such as efficient translation to automata and equivalence to first-order logic, as well as its interpretability, which arises from its resemblance to natural language.

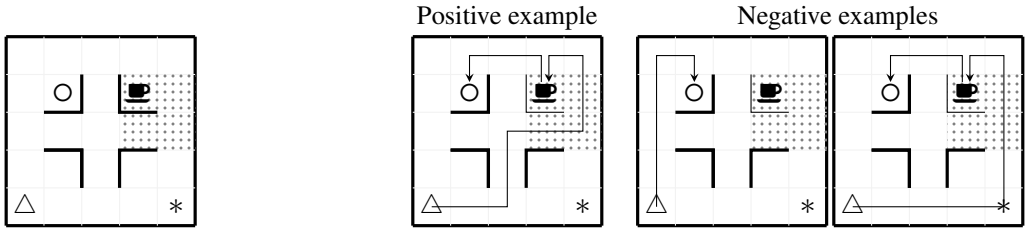
Traditionally, specifications (whether temporal or not) have been manually constructed. This approach is error-prone, time-consuming, and requires a detailed understanding of the underlying system [Bjørner and Havelund, 2014; Rozier, 2016]. As a result, in recent years, there has been concentrated effort on automatically designing reliable and interpretable specifications in temporal logics. A substantial body of research has centred on learning specifications in LTL [Neider and Gavran, 2018; Camacho and McIlraith, 2019; Raha *et al.*, 2022] and its continuous-time extension Signal Temporal Logic (STL) [Bombara *et al.*, 2016; Mohammadinejad *et al.*, 2020; Fronda and Abbas, 2022].

The primary setting of such learning frameworks is to infer specifications based on examples of trajectories generated from the underlying system. While these frameworks are effective in learning specifications for deterministic systems, a similar approach will not be sufficient for systems with stochastic behaviour. Specifications for these are inherently probabilistic, for example asserting that the probability of some behaviour being observed exceeds a given threshold. In these cases, it is not sufficient to infer a specification that characterises individual trajectories.

To accurately capture the behaviour of stochastic systems, we propose to learn temporal logic specifications from their formal models. As the logical specification formalism, we use *probabilistic LTL* (PLTL) [Vardi, 1985], which places thresholds on the probability of satisfaction of LTL formulas. The core models that we work with are discrete-time Markov chains (DTMCs). These are commonly used for modelling of stochastic systems in formal verification. In the context of RL, they capture the behaviour of an agent executing a specific learnt strategy (a.k.a., policy), in a probabilistic environment modelled as a Markov decision process (MDP).

We adopt the *passive learning* framework [Gold, 1978] and use a set of positive and negative DTMCs as input. The positive examples represent reliable probabilistic models or desirable strategies executing in stochastic environments, while the negative examples correspond to unreliable models or the behaviour of undesirable strategies. The learning task is to derive a concise PLTL specification that captures the probabilistic temporal behaviour exhibited by the positive DTMCs while excluding the behaviour of the negative DTMCs.

To illustrate our problem, we consider a simple stochastic office world environment, adapted from [Camacho *et al.*, 2019], shown in Figure 1. The environment consists of three features: office ○, coffee ☕, and decoration ∗. A slippery area (shaded) near the coffee station introduces randomness in the agent's movement. In this environment, desirable and undesirable strategies correspond to positive and negative DTMCs, respectively. Based on this input, a possible formula



(a) Office-world environment

(b) Strategies in the environment

Figure 1: An illustration of an office-world environment with the following features: office ○, coffee ☕, and decoration ∗. The shaded part near ☕ is slippery and introduces stochasticity in the agent's movement. The positive example demonstrates a strategy where the agent △ collects ☕ and delivers to ○ while avoiding ∗, whereas the negative examples do not achieve this temporal task.

could be $\mathbf{P}_{\geq 0.9}[\mathbf{F}(☕ \wedge \mathbf{F}(○))] \wedge \mathbf{P}_{\geq 0.9}[\mathbf{G}(\neg *)]$, which is a conjunction of two PLTL formulas. This formula uses the temporal operators $\mathbf{F}$ (*eventually*) and $\mathbf{G}$ (*always*), along with the probabilistic quantifier $\mathbf{P}_{\geq 0.9}$ (*at least 0.9 probability*) to state that the agent has a high probability of getting coffee and delivering it to the office while avoiding the decoration.

To address the passive learning problem for PLTL, we propose a novel symbolic search algorithm comprising three key procedures. The first procedure uses grammar-based enumeration to identify candidate LTL formulas. The second procedure determines threshold values for probabilistic quantifiers through probabilistic model checking, which results in a PLTL formula. Finally, the third procedure constructs Boolean combinations of PLTL formulas using a generalization of the set-cover problem.

To improve learning, the algorithm incorporates several heuristics for pruning the search space, including LTL simplification rules, inference techniques based on model checking, and tactics for Boolean combinations. Overall, the algorithm is designed with theoretical guarantees to learn concise and interpretable PLTL formulas from DTMCs.

We implement our algorithm as a tool `PriTL` to leverage the grammar-based search heuristic coupled with the state-of-the-art tool PRISM [Kwiatkowska *et al.*, 2011] for probabilistic model checking of DTMCs. We evaluate `PriTL` through two case studies. In the first case study, we consider learning PLTL formulas to distinguish between desirable and undesirable strategies obtained using RL for a variety of temporal tasks. In the second case study, we consider distinguishing between different variants of a probabilistic protocol. In both case studies, `PriTL` effectively infers concise and descriptive PLTL formulas that explain the probabilistic temporal behaviour of the systems. Moreover, we demonstrate how the different procedures contribute to the learning process. We refer the reader to the extended version [Roy *et al.*, 2025] for additional proofs, implementation details, and experimental results.

1.1 Related Work

There are two main areas of related work: learning temporal logics from data and explaining strategies/policies in RL.

Learning Temporal Logics. There are numerous works on learning temporal logics from data, specifically focusing on LTL [Neider and Roy, 2025] and STL [Bartocci *et al.*, 2022]. Our work falls within the category of *exact learning*, which seeks to infer minimal formulas that perfectly fit the data with provable guarantees. Notable works in this category include those for LTL [Neider and Gavran, 2018; Camacho and McIlraith, 2019; Raha *et al.*, 2022; Valizadeh *et al.*, 2024], STL [Mohammadinejad *et al.*, 2020], and several other temporal logics such as PSL [Roy *et al.*, 2020], CTL [Pommellet *et al.*, 2024] and ATL [Bordais *et al.*, 2024]. The learning techniques primarily involve deductive methods such as constraint solving and enumerative search.

There are also several works within the category of *approximate learning*, which seeks to infer formulas that fit (typically noisy) data well. Notable works in this category include those for LTL [Bartocci *et al.*, 2014; Luo *et al.*, 2022; Wan *et al.*, 2024; Chiariello, 2024] and STL [Nenzi *et al.*, 2018]. The learning techniques involve statistical optimisation, genetic algorithms, and neural network inference.

Our work considers the exact learning of probabilistic LTL, which, to our knowledge, has not been explored before. Moreover, we introduce a new learning framework based on symbolic search guided by dedicated model checkers.

Explaining RL policies Several approaches exist for explaining policies in reinforcement learning [Milani *et al.*, 2024]. Our work falls within the category of *global explanations* of pre-trained policies using formal languages. Notable works in this category include providing contrastive explanation using restricted queries in PCTL* [Boggess *et al.*, 2023], extracting finite-state machines from neural policies [Danesh *et al.*, 2021], and summarizing using abstract policy graphs [Topin and Veloso, 2019].

In contrast, our approach explains the temporal difference between policies using the full expressive power of PLTL. Such explanations can also be translated to natural language [Fuggitti and Chakraborti, 2023].

2 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \dots\}$ be the set of natural numbers.

2.1 Markov Chains and MDPs

A *discrete-time Markov chain* (DTMC) is a tuple $M = (S, s_I, P, \mathrm{AP}, \ell)$, where S is a finite set of states, $s_I \in S$ is an initial state, $P : S \times S \mapsto [0, 1]$ is a probabilistic transition function, AP is a set of atomic propositions and $\ell : S \mapsto 2^{\mathrm{AP}}$

is a labelling function. Atomic propositions will form the basis for temporal logic specifications and the function l defines the propositions that are true in each state.

A *path* π of M is an infinite sequence of states $\pi = s_0 s_1 s_2 \ldots \in S^\omega$ such that $P(s_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$. We denote the state at position i of a path π by $\pi[i] = s_i$ and the infinite suffix starting in $\pi[i]$ as $\pi[i:] = s_i s_{i+1} \ldots$. The set of all paths of M starting from state s is written as $\Pi^M(s)$. In standard fashion [Kemeny *et al.*, 1976], we define a probability measure $\Pr_s^M$ on the set of infinite paths $\Pi^M(s)$.

A *Markov decision process* (MDP) is a tuple $(S, s_I, A, P, \mathrm{AP}, \ell)$, which extends a DTMC by allowing a choice between actions in each state. The set of all actions is A and the probabilistic transition function becomes $P : S \times A \times S \mapsto [0, 1]$, where $P(s, a, s')$ is the probability to move to s' when action a is taken in s.

A *strategy* (a.k.a., policy) of an MDP defines which action is taken in each state, based on the history so far. In their most general form, strategies are defined as functions $\sigma : (S \times A)^* S \mapsto \Delta(A)$, where $\Delta(A)$ is the set of distributions over A. The behaviour of an MDP under a strategy σ is defined by an *induced DTMC*, which we denote by M^σ. In general, M^σ is infinite state. In this paper, however, we can restrict to finite-memory strategies, whose action choices depend only on the current state and a finite set of memory values, since these suffice for objectives specified in LTL. In this case, the induced DTMC M^σ is finite [Baier and Katoen, 2008].

2.2 Probabilistic Linear Temporal Logic (PLTL)

Probabilistic Linear Temporal Logic (PLTL) is the probabilistic variant of the popular logic LTL, and is commonly used to express the temporal behaviour of probabilistic systems. A PLTL formula takes the form $\mathbf{P}_{\bowtie p}[\varphi]$, stating that the probability with which LTL formula φ is satisfied meets the probability threshold $\bowtie p$. For example, PLTL formula $\mathbf{P}_{\geq 0.9}[\mathbf{F}(a \wedge \mathbf{F}\, b)]$ means that the probability of observing proposition a and then b is at least 0.9.

In this work, we learn specifications expressed in an extension of PLTL, which we call PLTL$^+$, that allows positive Boolean combinations of PLTL formulas.

Formally, the syntax and semantics of these logics are defined as follows. Firstly, LTL formulas φ are defined inductively using the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi,$$

where $p \in \mathrm{AP}$ is an proposition, $\neg$ (not), $\vee$ (or) and $\wedge$ (and) are standard Boolean operators, and $\mathbf{X}$ (neXt), $\mathbf{U}$ (Until) are standard temporal operators. We allow the standard temporal operators $\mathbf{F}$ (Finally) and $\mathbf{G}$ (Globally) as syntactic sugar, where $\mathbf{F}\,\varphi := \mathit{true}\,\mathbf{U}\,\varphi$ and $\mathbf{G}\,\varphi := \neg\,\mathbf{F}\,\neg\varphi$.

We interpret LTL formulas over paths of a DTMC. The satisfaction of LTL formula φ by (infinite) path π is defined inductively as follows:

$$\begin{aligned}
\pi &\models p \text{ iff } p \in \ell(\pi[0]) \\
\pi &\models \neg\varphi \text{ iff } \pi \not\models \varphi \\
\pi &\models \mathbf{X}\,\varphi \text{ iff } \pi[1:] \models \varphi \\
\pi &\models \varphi\,\mathbf{U}\,\varphi' \text{ iff there exists } i \in \mathbb{N} : \pi[i:] \models \varphi' \\
&\qquad\qquad \text{and for all } j < i : \pi[j:] \models \varphi
\end{aligned}$$

We interpret Boolean combinations in the standard fashion and therefore omit the definitions.

A PLTL$^+$ formula Φ is defined as:

$$\Phi ::= \mathbf{P}_{\bowtie p}[\varphi] \mid \Phi \vee \Phi \mid \Phi \wedge \Phi,$$

where $\bowtie \in \{<, >, \leq, \geq\}$, $p \in [0, 1]$ is a probability threshold and φ is an LTL formula.

A PLTL$^+$ (or PLTL) formula is interpreted over the states of a DTMC. The satisfaction of PLTL$^+$ formula Φ by a state s is defined as follows:

$$s \models \mathbf{P}_{\bowtie p}[\varphi] \text{ iff } \Pr^M(s \models \varphi) \bowtie p,$$

where $\Pr^M(s \models \varphi) = \Pr_s^M(\{\pi \in \Pi^M(s) \mid \pi \models \varphi\})$ denotes the probability that LTL formula φ is satisfied by a path starting in state s of M. We say that a DTMC M satisfies a PLTL$^+$ formula Φ if, for the initial state s_I of M, $s_I \models \Phi$.

3 Passive Learning of PLTL$^+$ Formulas

We frame the problem of learning a PLTL$^+$ formula as a typical *passive learning* problem [Gold, 1978]. Apart from being a fundamental learning problem, passive learning forms a key subroutine in other learning frameworks, such as active learning [Camacho and McIlraith, 2019] and learning from positive examples [Roy *et al.*, 2023].

In this problem, we rely on a *sample* $\mathcal{S} = (P, N)$ consisting of a set P of positive DTMCs and a set N of negative DTMCs. We define sample size $|\mathcal{S}|$ as the total number of DTMCs in $\mathcal{S}$. The goal is to learn a concise PLTL$^+$ formula Φ that is *consistent* with $\mathcal{S} = (P, N)$, i.e., for all $M \in P$, $M \models \Phi$, and for all $M \in N$, $M \not\models \Phi$.

To quantify conciseness, we measure the *size* $|\Phi|$ of PLTL$^+$ formulas Φ. To avoid checking redundant formulas, our learning algorithm uses LTL in *negation normal form* (NNF), a standard syntactic form where negation applies only to atomic propositions. We define the size of an LTL formula by the number of operators $\circ \in \{\mathbf{F}, \mathbf{X}, \mathbf{G}, \mathbf{U}, \wedge, \vee\}$ and literals $\Lambda = \{p, \neg p \mid p \in \mathrm{AP}\}$ in the formula. For instance, the formula $\mathbf{F}(p \wedge \mathbf{F}(\neg q))$ has size 5. The size of a PLTL$^+$ formula Φ is defined exactly the same way.

We now formally define the problem of passive learning of PLTL formulas.

Problem 1. *Given a sample $\mathcal{S} = (P, N)$, size bound K and propositions* AP*, learn a minimal PLTL$^+$ formula Φ over* AP *such that: (i) Φ is consistent with $\mathcal{S}$, and (ii) $|\Phi| \leq K$.*

A solution to Problem 1 (which is not necessarily unique) is a concise PLTL$^+$ formula Φ that distinguishes between the probabilistic temporal behaviour of the positive and negative DTMCs. The size bound K ensures three attributes for the solution formula Φ: (i) it does not get too large, (ii) it does not overfit to the sample, and (iii) it makes the passive learning problem decidable, ensuring a terminating algorithm.

4 The Learning Algorithm

We now describe the learning algorithm that we propose to solve Problem 1. Figure 2 provides a high-level overview of our algorithm. The algorithm consists of three main procedures: (i) *grammar-based enumeration*, which efficiently

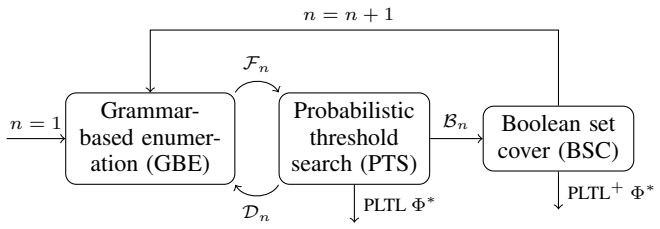


Figure 2: The high-level overview of the learning algorithm. The set $\mathcal{F}_n$ consists of formulas of size n, the set $\mathcal{D}_n$ consists of discarded formulas, and the set $\mathcal{B}_n$ consists of formulas for Boolean combinations. The procedures PTS and BSC output a consistent PLTL and PLTL$^+$ formula Φ^*, respectively, if they find one.

enumerates through the space of LTL formulas, (ii) *probabilistic threshold search*, which employs probabilistic model checking to determine whether a formula is consistent with the given sample, and (iii) *Boolean set cover*, which constructs Boolean combinations of PLTL formulas to form a consistent formula. The algorithm iterates over formulas of increasing size, starting from 1, using GBE, and then checks the consistency of the formulas using PTS and BSC. We now describe each of these procedures in detail.

4.1 Grammar-based Enumeration for LTL

The grammar-based enumeration (GBE) procedure incrementally explores the space of LTL formulas. Since the number of syntactically distinct formulas grows exponentially with formula size[1], GBE employs pruning techniques to manage the search space efficiently.

Importantly, GBE relies on the *nesting depth* (or depth, for brevity) of temporal operators in formulas. We define the nesting depth $d(\varphi)$ for LTL recursively as: $d(l) = 0$ for $l \in \Lambda$, $d(\varphi \circ \varphi') = \max(d(\varphi), d(\varphi'))$ for $\circ \in \{\wedge, \vee\}$, $d(\circ\varphi) = 1 + d(\varphi)$ for $\circ \in \{\mathbf{X}, \mathbf{F}, \mathbf{G}\}$, $d(\varphi\,\mathbf{U}\,\varphi') = 1+\max(d(\varphi), d(\varphi'))$. For instance, the formula $\mathbf{F}(p \wedge \mathbf{F}(\neg q))$ has depth 2.

The nesting depth is essential for managing the search space effectively as well as ensuring the practical applicability of the formulas. Heavily nested formulas (e.g., those with a depth > 3) are considered hard to interpret [Camacho and McIlraith, 2019] and are also uncommon in widely used LTL patterns [Dwyer *et al.*, 1998]. Therefore, GBE incorporates maximum depth D as a parameter.

To build formulas $\mathcal{F}_n$ of size n and all depths $d \leq D$, GBE employs a bottom-up dynamic programming approach. Specifically, the set $\mathcal{F}_n$ is built as a union of subsets $\mathcal{F}_n^d$ of LTL formulas of size n and depth d. GBE initializes $\mathcal{F}_1^0 := \Lambda$ and $\mathcal{F}_1^d := \emptyset$ for $0 < d \leq D$. It then inductively combines formulas from $\mathcal{F}_n^d$ of different depths to form larger formulas.

The inductive step of GBE $\mathcal{F}_{n+1}$ is outlined in Algorithm 1. It follows the LTL grammar, adding operators to smaller formulas to construct larger ones. The algorithm uses two heuristics to eliminate semantically equivalent ($\equiv$) formulas, where $\varphi \equiv \varphi'$ if and only if $\pi \models \varphi \leftrightarrow \pi \models \varphi'$ for any path $\pi \in (2^{\mathrm{AP}})^\omega$. These heuristics are briefly described here.

[1]Asymptotically $\frac{7^n\sqrt{14}}{2\sqrt{\pi n^3}}$ [Flajolet and Sedgewick, 2009]

Algorithm 1 Inductive step in GBE

Input: $\mathcal{F}_n$, Max depth D

for $d = 0$ to D **do**
  $\mathcal{F}_{n+1}^d = \emptyset$
  for $\varphi \in \mathcal{F}_n^{d-1}$ **do**
    Construct $\psi = \circ\varphi$ for $\circ \in \{\mathbf{X}, \mathbf{F}, \mathbf{G}\}$
    Add ψ to $\mathcal{F}_{n+1}^d$ if *temporal simplify* does not hold
  end for
  for $k = 1$ to $n - 1$ **do**
    for $\varphi \in \mathcal{F}_k^{d-1}$ and $\varphi' \in \bigcup_{d'<d} \mathcal{F}_{n-k}^{d'}$ **do**
      $\psi = \varphi\,\mathbf{U}\,\varphi'$
      Add ψ to $\mathcal{F}_{n+1}^d$ if *Boolean simplify* does not hold
    end for
    for $\varphi \in \mathcal{F}_k^d$ and $\varphi' \in \bigcup_{d'\leq d} \mathcal{F}_{n-k}^{d'}$ **do**
      $\psi = \varphi \circ \varphi'$ for $\circ \in \{\wedge, \vee\}$
      Add ψ to $\mathcal{F}_{n+1}^d$ if *Boolean simplify* does not hold
    end for
  end for
end for
return $\mathcal{F}_{n+1}$

The first heuristic, *temporal simplification*, removes redundant formulas by applying syntactic rules to rewrite LTL formulas into normal forms [Baier and Katoen, 2008, Fig. 5.7]. For example, $\mathbf{F}\,\mathbf{F}(p) \equiv \mathbf{F}(p)$, $\mathbf{F}\,\mathbf{X}(p) \equiv \mathbf{X}\,\mathbf{F}(p)$, and $\mathbf{F}\,\mathbf{G}\,\mathbf{F}(p) \equiv \mathbf{G}\,\mathbf{F}(p)$ (see [Duret-Lutz, 2024, Sec. 5.4] for the full list). If a constructed formula is not in simplified form, it is discarded. These checks are constant-time operations, making them highly efficient.

The second heuristic, *Boolean simplification*, removes redundant Boolean combinations such as $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where $\varphi_1 \equiv \varphi_2$ or $\varphi_1 \equiv \neg\varphi_2$. It checks the syntactic equality of φ_1 and φ_2 via a linear-time scan of their syntax trees. It then checks semantic equivalence, which can be doubly exponential in formula size but is efficiently handled by modern LTL satisfiability checkers [Duret-Lutz *et al.*, 2022].

We formalise the completeness of the GBE procedure below[2], which can be proved by induction on formula size.

Lemma 1. *$\bigcup_{n\leq N'} \mathcal{F}_n$ computed by GBE consists of all semantically distinct formulas of size $\leq N'$ and depth $\leq D$.*

4.2 Probabilistic Threshold Search for PLTL

The probabilistic threshold search (PTS) procedure steps through the formulas generated by GBE and evaluates the likelihood of the formulas being satisfied/consistent. The main steps of PTS are outlined in Algorithm 2.

PTS first computes the probability measure for a formula $\varphi \in \mathcal{F}_n$ for each M in the given sample. More specifically, PTS computes the vector $V^{M,\varphi} : S \to [0, 1]$, mapping each state $s \in S$ of M to $\Pr^M(s \models \varphi)$. We use $v_I^{M,\varphi} = V^{M,\varphi}(s_I)$ to denote the probability for the initial state s_I of M.

Our implementation, which is based on the PRISM tool [Kwiatkowska *et al.*, 2011], deploys standard proba-

[2]This result establishes GBE's complete search in isolation; when combined with PTS, as we see later in Section 4.2, several formulas are pruned using heuristics.

bilistic LTL model checking procedures [Baier and Katoen, 2008]. First, the LTL formula φ is translated into an equivalent deterministic Rabin automaton (DRA) $\mathcal{A}_\varphi$. Then the product DTMC $M \times \mathcal{A}_\varphi$, which combines the DTMC and DRA, is constructed and solved using standard numerical methods based on value iteration.

PTS exploits the computed vector $V^{M,\varphi}$ to search for a formula that has a higher probability of satisfaction in the positive examples than in the negative examples. For this, it computes the minimum probability $p_\varphi = \min\{v_I^{M,\varphi} \mid M \in P\}$ of satisfaction of φ among the samples in the set P, and the maximum probability $n_\varphi = \max\{v_I^{M,\varphi} \mid M \in N\}$ of satisfaction of φ among the samples in the set N.

To identify a significant probabilistic difference in the temporal behaviour, PTS employs a small tolerance parameter $\delta \in (0, 0.1)$. We explain the motivation for this parameter using the introductory example from Figure 1. In this example, the probability of reaching the office, i.e., satisfying $\mathbf{F}(\bigcirc)$, can be slightly lower in the positive example (say 0.94) compared to the negative examples (say, 1.0 and 0.95). This small difference could arise because the agent takes a slightly longer slippery route in the positive example than in the second negative example. However, $\mathbf{F}(\bigcirc)$ is not a primary distinguishing factor between the positive and negative examples and must not be considered by PTS.

Thus, PTS checks if the difference $p_\varphi - n_\varphi$ between the probability of satisfaction of φ in the positive and negative examples is greater than the tolerance δ. If indeed $p_\varphi - n_\varphi > \delta$, then PTS outputs the formula $\Phi = \mathbf{P}_{>m_\varphi}[\varphi]$, where the threshold $m_\varphi = \frac{p_\varphi + n_\varphi}{2}$. While any threshold between p_φ and n_φ could be chosen, the choice of the mean of the two values is to reduce overfitting to the input sample.

We state the soundness of the PTS procedure as follows.

Lemma 2. *If PTS returns a PLTL formula Φ, then Φ is consistent with sample $\mathcal{S}$.*

Proof. PTS always returns a formula of the form $\Phi = \mathbf{P}_{>m_\varphi}[\varphi]$, where $m_\varphi = \frac{p_\varphi + n_\varphi}{2}$ and $p_\varphi - n_\varphi > \delta$. We can state that

$$\forall M \in P, v_I^{M,\varphi} > m_\varphi \text{ iff } \Pr^M(s_I \models \varphi) > m_\varphi \text{ iff } M \models \Phi,$$
$$\forall M \in N, v_I^{M,\varphi} < m_\varphi \text{ iff } \Pr^M(s_I \models \varphi) < m_\varphi \text{ iff } M \not\models \Phi.$$

□

Note that PTS restricts the search to only PLTL formulas of the form $\mathbf{P}_{>p}[\varphi]$. The relation $\geq$ is not required due to the non-zero parameter δ, while the $<$ relation can be derived from the $>$ relation and the dual LTL formula $\neg\varphi$ using the relation $\mathbf{P}_{<p}[\varphi] \equiv \mathbf{P}_{>1-p}[\neg\varphi]$.

If a formula φ is not consistent, PTS discards it by adding it to $\mathcal{D}_n$ if it is not useful for the next GBE iterations; otherwise, it adds φ to $\mathcal{B}_n$ for Boolean combinations. We briefly discuss the heuristics used for discarding formulas.

This heuristic, *inconsistency removal*, discards φ if the following condition holds: $V^{M,\varphi} \equiv \mathbf{0}$ for each $M \in P$, or $V^{M,\varphi} \equiv \mathbf{1}$ for each $M \in N$, where $\mathbf{0}$ and $\mathbf{1}$ are the vectors with all zeros and all ones, respectively. In simpler terms, φ is discarded if it is unsatisfiable in any state of the positive DTMCs or universally satisfied in all states of the negative DTMCs. Such formulas cannot be meaningfully combined in subsequent iterations, as stated below for the positive cases; a similar argument applies to the negative cases.

Algorithm 2 Probabilistic Threshold Search (PTS)

Input: $\mathcal{F}_n$, Probabilistic tolerance δ

for $d = 0$ to D, $\varphi \in \mathcal{F}_n^d$ **do**
  Compute $V^{M,\varphi}$ for each M in $\mathcal{S}$
  $p_\varphi = \min_{M \in P}\{v_I^{M,\varphi}\}$, $n_\varphi = \max_{M \in N}\{v_I^{M,\varphi}\}$
  if $p_\varphi - n_\varphi > \delta$ **then**
    return $\Phi = \mathbf{P}_{> \frac{p_\varphi + n_\varphi}{2}}[\varphi]$
  else
    Add φ to $\mathcal{D}_n$ if *inconsistency removal* holds
    Add φ to $\mathcal{B}_n$ otherwise
  end if
end for

Lemma 3. *Let $V^{M,\varphi} \equiv \mathbf{0}$ for each $M \in P$. Then φ cannot be a subformula of a minimal consistent PLTL formula.*

Examples of formulas that can be discarded from the introductory example include $\mathbf{F}(* \wedge \text{☕})$, $\mathbf{G}(\text{☕})$, and $\mathbf{G}(*)$ since they never hold in any state of the positive DTMC.

4.3 Boolean Set Cover for PLTL$^+$

The Boolean Set Cover (BSC) procedure combines PLTL formulas using Boolean operations. We adapt this procedure, originally introduced in [Raha *et al.*, 2022], to accommodate probability thresholds. The steps of our algorithm are detailed in Algorithm 3.

First, BSC discards formulas from $\mathcal{B}_n$ that are not useful for Boolean combinations. For this, it uses a condition similar to, but weaker than, inconsistency removal used in PTS: $v_I^{M,\varphi} = 0$ for all $M \in P$, or $v_I^{M,\varphi} = 1$ for all $M \in N$.

For the remaining formulas, BSC assesses how close they are to being a consistent formula. To do this, it relies on the function $c(\varphi, r)$, which quantifies the quality of φ with probability threshold r, defined as follows:

$$c(\varphi, r) = \Big[\sum_{M \in P} [\![v_I^{M,\varphi} > r]\!] + \sum_{M \in N} [\![v_I^{M,\varphi} < r]\!] \Big],$$

where $[\![\cdot]\!]$ denotes the Iverson bracket, evaluating to 1 if the condition holds, and 0 otherwise. We have $c(\varphi, r) = |\mathcal{S}|$ if and only if $\mathbf{P}_{>r}[\varphi]$ is consistent with $\mathcal{S}$.

BSC computes, for each LTL formula $\varphi \in \mathcal{B}_n$, a maximal probability threshold $r^* = \arg\max_{r \in (0,1)} c(\varphi, r)$ that maximizes consistency with $\mathcal{S}$. This can be computed via a linear scan over the sorted list of probabilities $v_I^{M,\varphi}$ for M in $\mathcal{S}$.

BSC then constructs the PLTL formula $\Phi = \mathbf{P}_{>r^*}[\varphi]$ along with its score $\sigma(\Phi) = c(\varphi, r^*)/(1 + \sqrt{|\Phi|})$ and adds it to a heap $\mathcal{H}$. The scoring function and the subsequent steps of BSC are as in [Raha *et al.*, 2022]. Briefly, a maximum Boolean combination limit L is considered. The PLTL formulas with the L highest scores are selected, and combined as disjunctions and conjunctions with all formulas in $\mathcal{H}$.

Below, we state the soundness of BSC for PLTL, based on [Raha *et al.*, 2022].

Algorithm 3 Boolean Set Cover (BSC) for PLTL

Input: $\mathcal{B}_n$, Max size K, Max Limit L

Discard formulas from $\mathcal{B}_n$ not suitable for bool comb
for $\varphi \in \mathcal{B}_n$ **do**
  Compute $\Phi = \mathbf{P}_{>r^*}[\varphi]$, score $\sigma(\Phi)$ and add to $\mathcal{H}$
end for
$\mathcal{H}^* \leftarrow$ Highest L formulas in $\mathcal{H}$ w.r.t score σ
for $\Psi \in \mathcal{H}$ and $\Phi \in \mathcal{H}^*$ **do**
  $\Phi' := \Psi \circ \Phi$ for $\circ \in \{\wedge, \vee\}$
  if $|\Phi'| \leq K$ and Φ' is consistent **then**
    Store Φ' as consistent and update $K \leftarrow |\Phi'| - 1$
  end if
end for

Lemma 4. *If BSC returns a PLTL$^+$ formula Φ, then Φ is consistent with sample $\mathcal{S}$.*

Theoretical guarantees. We state the guarantees of our algorithm with respect to the search space $\Theta(K, D, \delta)$ of PLTL formulas constrained by the considered parameters, i.e., size $\leq K$, depth $\leq D$ and tolerance $> \delta$.

Theorem 1. *Given sample $\mathcal{S}$, size K, depth D, and tolerance δ, our learning algorithm has the following guarantees:*

- *(soundness) if it returns a PLTL$^+$ formula Φ, then Φ is consistent with $\mathcal{S}$ and $|\Phi| \leq K$, and*
- *(completeness and minimality) if there exists a PLTL formula in $\Theta(K, D, \delta)$ consistent with $\mathcal{S}$, then it returns a minimal PLTL$^+$ formula.*

Proof. Soundness follows from the correctness of PTS (Lemma 2) and BSC (Lemma 4) in outputting a consistent formula. Completeness is ensured by the exhaustive enumeration by GBE (Lemma 1), discarding only inconsistent formulas (Lemma 3). Minimality follows from the complete iterative search over increasing formula sizes (see Fig. 2). □

5 Evaluation

In this section, we evaluate the capability of our learning algorithm to infer concise PLTL$^+$ formulas from samples of DTMCs. To this end, we developed a prototype tool, `PriTL`[3], implemented in Python3, which integrates the three procedures, GBE, PTS and BSC, of the learning algorithm. For heuristics in GBE, we rely on LTL simplification and satisfaction features from the SPOT library [Duret-Lutz *et al.*, 2022]. For LTL model checking of DTMCs in PTS, we rely on the PRISM tool [Kwiatkowska *et al.*, 2011], using its (default) hybrid model checking engine.

To the best of our knowledge, no existing tool can directly learn arbitrary temporal specifications from DTMCs. To evaluate `PriTL`'s ability to learn concise and distinguishing formulas, we tested it on strategies generated within a stochastic environment and on various variants of a probabilistic model. For all experiments, we set the maximum depth $D = 2$, tolerance $\delta = 0.05$, and a Boolean combination limit $L = 10$. If `PriTL` identifies multiple minimal formulas, it returns the one with the highest probability difference, $p_\varphi - n_\varphi$. In case of a tie, `PriTL` returns all valid formulas.

We conducted all experiments on a MacBook Pro M3 (macOS 14.6.1) with 18 GB RAM. A detailed discussion of the implementation and experiments can be found in the appendix of the extended version [Roy *et al.*, 2025].

Learning from strategies in stochastic environment

For this experiment, we focus on strategy DTMCs generated via non-Markovian reinforcement learning algorithms. Specifically, we utilize different Q-learning algorithms proposed by [Shao and Kwiatkowska, 2023] that are capable of generating optimal strategies for LTL tasks.

As the underlying MDP, we select the widely used OpenAI Gym frozen lake environment [Brockman *et al.*, 2016], employing the same layout as in [Shao and Kwiatkowska, 2023]. This environment is an 8×8 gridworld, where an agent navigates a slippery frozen lake, introducing stochasticity: with a 1/3 probability, the agent moves in the intended direction, and with a 1/3 probability, it deviates sideways. The environment includes three key features: two campsites, a and b, and several holes h.

We evaluate `PriTL` on two distinct applications: (i) learning from strategies trained on correct and incorrect LTL tasks, and (ii) learning from optimal and suboptimal strategies for the same LTL task. For both (i) and (ii), we set the propositions $\mathrm{AP} = \{a, b, h\}$ and formula size bound $K = 10$.

For application (i), we identify several desirable LTL tasks and designate them as correct tasks. As incorrect tasks, we select LTL tasks that are less precise than their correct counterparts. For example, the correct task $\mathbf{F}(a) \wedge \mathbf{G}(\neg h)$ requires reaching campsite a while always avoiding holes h, whereas the incorrect task $\mathbf{F}(a)$ specifies a weaker condition of reaching the campsite, which may result in falling into holes. The first two columns of Table 1 list the considered correct and incorrect tasks, respectively. For each correct and incorrect task, we generate 10 positive and 10 negative optimal strategy DTMCs, respectively, using the CF+KC Q-learning algorithm [Shao and Kwiatkowska, 2023], known for its fast convergence to optimality. The cumulative state space of the samples is of the order of 10^3 (see fourth column of Table 1).

We present the learned PLTL$^+$ formulas for each task in Table 1. For the first two tasks, `PriTL` inferred PLTL$^+$ formulas with safety properties, $\mathbf{G}(\neg h)$ and $\neg h\, \mathbf{U}\, a$, which were violated in the negative examples. In subsequent tasks, `PriTL` inferred formulas that indicate the specific requirements missing in the negative examples. These include repeated reachability $\mathbf{G}\,\mathbf{F}(a)$ instead of simple reachability $\mathbf{F}(a)$, performing two tasks simultaneously $\mathbf{F}(a) \wedge \mathbf{F}(b)$ instead of just one $\mathbf{F}(a)$ or $\mathbf{F}(b)$, etc. Overall, `PriTL` successfully produced concise formulas that explain the differences between strategies trained on different tasks.

For application (ii), we identify some more desirable LTL tasks (in Figure 3) and generate optimal and suboptimal strategies for each. We extract strategy DTMCs from intermediate episodes of the KC Q-learning algorithm since it has relatively slower convergence to optimality [Shao and Kwiatkowska, 2023], thereby often yielding sub-optimal strategies. We considered a strategy that achieves a high prob-

[3]https://github.com/rajarshi008/PriTL

Correct task for P	Incorrect task(s) for N	Learned PLTL$^+$ Formula	State space $\mathcal{S}$	LTL Space	Time (sec)
$\mathbf{F}(a) \wedge \mathbf{G}(\neg h)$	$\mathbf{F}(a)$	$\mathbf{P}_{>0.76}[\mathbf{G}(\neg h)]$	$2.5 \cdot 10^3$	24/24	1.77
$\mathbf{F}(a) \wedge \mathbf{G}(\neg h)$	$\mathbf{F}(a)$, $\mathbf{G}(\neg h)$	$\mathbf{P}_{>0.76}[\neg h \,\mathbf{U}\, a]$	$1.9 \cdot 10^3$	110/186	3.39
$\mathbf{F}\,\mathbf{G}(a) \wedge \mathbf{G}(\neg h)$	$\mathbf{F}(a) \wedge \mathbf{G}(\neg h)$	$\mathbf{P}_{>0.49}[\mathbf{F}\,\mathbf{G}(a)]$, $\mathbf{P}_{>0.49}[\mathbf{G}\,\mathbf{F}(a)]$	$2.7 \cdot 10^3$	112/186	4.93
$\mathbf{G}\,\mathbf{F}(a) \wedge \mathbf{G}\,\mathbf{F}(\neg a) \wedge \mathbf{G}(\neg h)$	$\mathbf{F}\,\mathbf{G}(a) \wedge \mathbf{G}(\neg h)$	$\mathbf{P}_{>0.5}[\mathbf{G}\,\mathbf{F}(\neg a)]$	$0.9 \cdot 10^3$	50/186	3.9
$\mathbf{F}(a) \wedge \mathbf{F}(b) \wedge \mathbf{G}(\neg h)$	$\mathbf{F}(a) \wedge \mathbf{G}(\neg h)$, $\mathbf{F}(b) \wedge \mathbf{G}(\neg h)$	$\mathbf{P}_{>0.5}[\mathbf{F}(a)] \wedge \mathbf{P}_{>0.99}[\mathbf{F}(b)]$	$3.2 \cdot 10^3$	476/7314	24.1

Table 1: Summary of the learning from strategy DTMCs for correct and incorrect tasks on Frozen Lake.

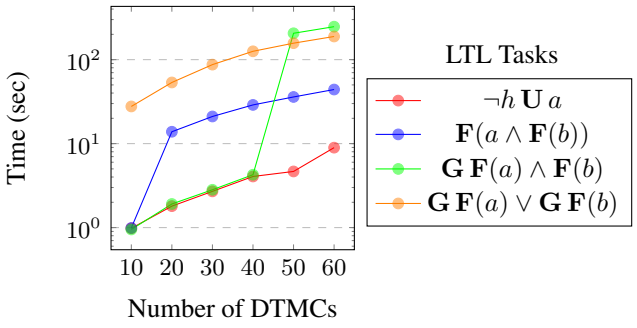


Figure 3: Runtime comparison for strategies generated from varying formulas and varying sample sizes.

ability (i.e., $p \geq 0.95$) as optimal, while one that achieves a lower probability (i.e., $0.5 \leq p \leq 0.9$) as suboptimal. Overall, for each LTL task, we collected at least 30 positive DTMCs and 30 negative DTMCs corresponding to optimal and suboptimal strategies, respectively. We evaluated our algorithm using varying sample sizes $|\mathcal{S}|$, ranging from 10 to 60 DTMCs per sample, with an equal split between positive and negative examples.

We present the runtime for varying sample sizes in Figure 3. For the tasks $\neg h \,\mathbf{U}\, a$ and $\mathbf{G}\,\mathbf{F}(a) \vee \mathbf{G}\,\mathbf{F}(b)$, `PriTL` consistently inferred the formulas $\mathbf{P}_{>0.9}[\mathbf{G}(\neg h)]$ and $\mathbf{P}_{>0.93}[\mathbf{G}\,\mathbf{F}(a) \vee \mathbf{G}\,\mathbf{F}(b)]$, respectively, across all samples. The runtime for these increased linearly with the sample size.

For the task $\mathbf{F}(a \wedge \mathbf{F}(b))$, `PriTL` inferred $\mathbf{P}_{>0.97}[\mathbf{F}(b)]$ for a sample size of 10 and a more precise formula, $\mathbf{P}_{>0.94}[\mathbf{F}(a)] \wedge \mathbf{P}_{>0.97}[\mathbf{F}(b)]$, for larger sizes. Similarly, for the task $\mathbf{G}\,\mathbf{F}(a) \wedge \mathbf{F}(b)$, `PriTL` inferred $\mathbf{P}_{>0.92}[\mathbf{F}(b)]$, for smaller samples ($\leq 40$), whereas it inferred a more precise formula, $\mathbf{P}_{>0.92}[\mathbf{G}\,\mathbf{F}(a)] \wedge \mathbf{P}_{>0.95}[\mathbf{F}(b)]$ for larger samples (50 and 60). Both tasks showed a runtime spike due to the change in the inferred formula, deviating from the linear trend in other tasks. Moreover, in both cases, the more precise formula was inferred by the BSC procedure. Overall, `PriTL` successfully inferred expected PLTL$^+$ formulas, with runtime generally scaling linearly with sample size.

We also briefly discuss how different parts of `PriTL` contribute to the learning process. PTS dominates the running time, while GBE and BSC take negligible time. For example, inferring $\mathbf{P}_{>0.5}[\mathbf{F}(a)] \wedge \mathbf{P}_{>0.99}[\mathbf{F}(b)]$ (from Table 1) took 24.05 seconds for PTS, and 0.05 and 0.01 seconds for GBE and BSC, respectively. The heuristics enhance efficiency by reducing the search space, particularly for larger formulas. The fifth column of Table 1 compares the considered formula for PTS with the total space up to the size of the learned formula. For the same example, the search was reduced to just 7% of the possible LTL space via heuristics.

Learning from variants of probabilistic models

In this experiment, we compare two implementations of the probabilistic secret-sharing protocol EGL [Even *et al.*, 1985; Norman and Shmatikov, 2006], where two parties, A and B, share $2P$ secrets (2-length bit-vectors) over several rounds. The two implementations we consider, EGL1$_P$ and EGL2$_P$, are parametrized by the number of secrets $P = 1, \ldots, 7$. The key difference in the variants is in the sharing order: in EGL1$_P$, each party sequentially shares all their i^{th} bits, while in EGL2$_P$, they alternately share half of their i^{th} bits.

To apply `PriTL`, we treat EGL1$_P$ as positive, EGL2$_P$ as negative, $K = 6$ and AP $= \{\texttt{kA}, \texttt{kB}\}$, where `kA` (`kB` resp.) represents A (B resp.) knows of B's (A's resp.) secrets.

For $P = 1, 2, 3, 4, 5$, `PriTL` inferred the formula $\mathbf{P}_{>p}[\neg \texttt{kA} \,\mathbf{U}\, \texttt{kB}]$ with progressively increasing thresholds $p = 0.88, 0.90, 0.94, 0.97$ with running times 0.37, 0.38, 0.68, 1.31 seconds, respectively. The inferred formula indicates that, in EGL1$_P$, the probability that B knows A's secret before A knows B's is higher as compared to EGL2$_P$.

For $P > 5$, however, `PriTL` did not infer any formula. This indicates that no PLTL formula with parameters $K \leq 6$, $d \leq 2$ and $\delta > 0.05$ distinguishes the variants when a higher number of secrets are shared, based on our exhaustive search (Theorem 1). Overall, `PriTL` could identify key probabilistic temporal differences between variants of probabilistic models, or confirm their absence.

6 Conclusion

We focused on the automatic learning of temporal behaviour in stochastic systems. Specifically, we considered the passive learning problem of learning concise probabilistic LTL (PLTL) formulas that distinguish between positive and negative Markov chains. Our novel learning algorithm combines grammar-based enumeration with probabilistic model checking, enhanced by search heuristics. We demonstrated the ability of our approach in inferring temporal specifications in both reinforcement learning and modelling applications.

In the future, we plan to integrate our algorithm into other learning frameworks, such as active learning [Camacho and McIlraith, 2019] and learning from positive examples [Roy *et al.*, 2023]. Moreover, we aim to extend our approach to multi-agent systems [Boggess *et al.*, 2023].

Acknowledgements

This project received funding from the ERC under the European Union's Horizon 2020 research and innovation programme (grant agreement No.834115, FUN2MODEL). The work was done while YP visited the University of Oxford for an internship.

References

[Baier and Katoen, 2008] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[Bartocci *et al.*, 2014] Ezio Bartocci, Luca Bortolussi, and Guido Sanguinetti. Data-driven statistical learning of temporal logic properties. In *FORMATS*, volume 8711 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 2014.

[Bartocci *et al.*, 2022] Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Nickovic. Survey on mining signal temporal logic specifications. *Inf. Comput.*, 289(Part):104957, 2022.

[Bjørner and Havelund, 2014] Dines Bjørner and Klaus Havelund. 40 years of formal methods - some obstacles and some possibilities? In *FM*, volume 8442 of *Lecture Notes in Computer Science*, pages 42–61. Springer, 2014.

[Boggess *et al.*, 2023] Kayla Boggess, Sarit Kraus, and Lu Feng. Explainable multi-agent reinforcement learning for temporal queries. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 55–63. ijcai.org, 2023.

[Bombara *et al.*, 2016] Giuseppe Bombara, Cristian Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. A decision tree approach to data classification using signal temporal logic. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, HSCC '16, page 1–10, New York, NY, USA, 2016. Association for Computing Machinery.

[Bordais *et al.*, 2024] Benjamin Bordais, Daniel Neider, and Rajarshi Roy. Learning branching-time properties in CTL and ATL via constraint solving. In *Formal Methods - 26th International Symposium, FM 2024, Milan, Italy, September 9-13, 2024, Proceedings, Part I*, volume 14933 of *Lecture Notes in Computer Science*, pages 304–323. Springer, 2024.

[Bozkurt *et al.*, 2020] Alper Kamil Bozkurt, Yu Wang, Michael M. Zavlanos, and Miroslav Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 10349–10355. IEEE, 2020.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[Camacho and McIlraith, 2019] Alberto Camacho and Sheila A. McIlraith. Learning interpretable models expressed in linear temporal logic. In *ICAPS*, pages 621–630. AAAI Press, 2019.

[Camacho *et al.*, 2019] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, pages 6065–6073. ijcai.org, 2019.

[Chiariello, 2024] Francesco Chiariello. Learning temporal properties from event logs via sequential analysis. In *TIME*, volume 318 of *LIPIcs*, pages 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[Danesh *et al.*, 2021] Mohamad H. Danesh, Anurag Koul, Alan Fern, and Saeed Khorram. Re-understanding finite-state representations of recurrent policy networks. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 2388–2397. PMLR, 2021.

[Duret-Lutz *et al.*, 2022] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer, August 2022.

[Duret-Lutz, 2024] Alexandre Duret-Lutz. Spot's temporal logic formulas, 2024. https://spot.lre.epita.fr/tl.pdf, Accessed: 02-01-2025.

[Dwyer *et al.*, 1998] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *FMSP*, pages 7–15. ACM, 1998.

[Even *et al.*, 1985] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[Flajolet and Sedgewick, 2009] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

[Fronda and Abbas, 2022] Nicole Fronda and Houssam Abbas. Differentiable inference of temporal logic formulas. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 41(11):4193–4204, 2022.

[Fuggitti and Chakraborti, 2023] Francesco Fuggitti and Tathagata Chakraborti. NL2LTL - a python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas. In *AAAI*, pages 16428–16430. AAAI Press, 2023.

[Gold, 1978] E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3):302–320, 1978.

[Hasanbeig *et al.*, 2019] Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J. Pappas, and Insup Lee. Reinforcement learning

for temporal logic control synthesis with probabilistic satisfaction guarantees. In *58th IEEE Conference on Decision and Control, CDC 2019, Nice, France, December 11-13, 2019*, pages 5338–5343. IEEE, 2019.

[Kemeny *et al.*, 1976] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer-Verlag, 2nd edition, 1976.

[Kwiatkowska *et al.*, 2011] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[Li *et al.*, 2017] Xiao Li, Cristian Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 3834–3839. IEEE, 2017.

[Luo *et al.*, 2022] Weilin Luo, Pingjia Liang, Jianfeng Du, Hai Wan, Bo Peng, and Delong Zhang. Bridging ltlf inference to GNN inference for learning ltlf formulae. In *AAAI*, pages 9849–9857. AAAI Press, 2022.

[Milani *et al.*, 2024] Stephanie Milani, Nicholay Topin, Manuela Veloso, and Fei Fang. Explainable reinforcement learning: A survey and comparative review. *ACM Comput. Surv.*, 56(7):168:1–168:36, 2024.

[Mohammadinejad *et al.*, 2020] Sara Mohammadinejad, Jyotirmoy V. Deshmukh, Aniruddh Gopinath Puranic, Marcell Vazquez-Chanlatte, and Alexandre Donzé. Interpretable classification of time-series data using efficient enumerative techniques. In *HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020*, pages 9:1–9:10. ACM, 2020.

[Neider and Gavran, 2018] Daniel Neider and Ivan Gavran. Learning linear temporal properties. In *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*, pages 1–10. IEEE, 2018.

[Neider and Roy, 2025] Daniel Neider and Rajarshi Roy. What is formal verification without specifications? A survey on mining LTL specifications. In *Principles of Verification (3)*, volume 15262 of *Lecture Notes in Computer Science*, pages 109–125. Springer, 2025.

[Nenzi *et al.*, 2018] Laura Nenzi, Simone Silvetti, Ezio Bartocci, and Luca Bortolussi. A robust genetic algorithm for learning temporal specifications from data. In *QEST*, volume 11024 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2018.

[Norman and Shmatikov, 2006] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[Pommellet *et al.*, 2024] Adrien Pommellet, Daniel Stan, and Simon Scatton. Sat-based learning of computation tree logic. In *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I*, volume 14739 of *Lecture Notes in Computer Science*, pages 366–385. Springer, 2024.

[Raha *et al.*, 2022] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, and Daniel Neider. Scalable anytime algorithms for learning fragments of linear temporal logic. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 263–280, Cham, 2022. Springer International Publishing.

[Roy *et al.*, 2020] Rajarshi Roy, Dana Fisman, and Daniel Neider. Learning interpretable models in the property specification language. In *IJCAI*, pages 2213–2219. ijcai.org, 2020.

[Roy *et al.*, 2023] Rajarshi Roy, Jean-Raphaël Gaglione, Nasim Baharisangari, Daniel Neider, Zhe Xu, and Ufuk Topcu. Learning interpretable temporal properties from positive examples only. In *AAAI*, pages 6507–6515. AAAI Press, 2023.

[Roy *et al.*, 2025] Rajarshi Roy, Yash Pote, David Parker, and Marta Kwiatkowska. Learning probabilistic temporal logic specifications for stochastic systems. *CoRR*, abs/2505.12107, 2025.

[Rozier, 2016] Kristin Yvonne Rozier. Specification: The biggest bottleneck in formal methods and autonomy. In *VSTTE*, volume 9971 of *Lecture Notes in Computer Science*, pages 8–26, 2016.

[Shao and Kwiatkowska, 2023] Daqian Shao and Marta Kwiatkowska. Sample efficient model-free reinforcement learning from LTL specifications with optimality guarantees. In *IJCAI*, pages 4180–4189. ijcai.org, 2023.

[Topin and Veloso, 2019] Nicholay Topin and Manuela Veloso. Generation of policy-level explanations for reinforcement learning. In *AAAI*, pages 2514–2521. AAAI Press, 2019.

[Valizadeh *et al.*, 2024] Mojtaba Valizadeh, Nathanaël Fijalkow, and Martin Berger. Ltl learning on gpus. In *Computer Aided Verification*, pages 209–231, Cham, 2024. Springer Nature Switzerland.

[Vardi, 1985] M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society Press, 1985.

[Wan *et al.*, 2024] Hai Wan, Pingjia Liang, Jianfeng Du, Weilin Luo, Rongzhen Ye, and Bo Peng. End-to-end learning of ltlf formulae by faithful ltlf encoding. In *AAAI*, pages 9071–9079. AAAI Press, 2024.