Planning with Uncertain Specifications (PUnS)

Ankit Shah[1], Shen Li[2], Julie Shah[3]

***Abstract*—Reward engineering is crucial to high performance in reinforcement learning systems. Prior research into reward design has largely focused on Markovian functions representing the reward. While there has been research into expressing non-Markov rewards as linear temporal logic (LTL) formulas, this has focused on task specifications directly defined by the user. However, in many real-world applications, task specifications are ambiguous, and can only be expressed as a belief over LTL formulas. In this paper, we introduce planning with uncertain specifications (PUnS), a novel formulation that addresses the challenge posed by non-Markovian specifications expressed as beliefs over LTL formulas. We present four criteria that capture the semantics of satisfying a belief over specifications for different applications, and analyze the qualitative implications of these criteria within a synthetic domain. We demonstrate the existence of an equivalent Markov decision process (MDP) for any instance of PUnS. Finally, we demonstrate our approach on the real-world task of setting a dinner table automatically with a robot that inferred task specifications from human demonstrations.**

***Index Terms*—AI-Based Methods, Learning from Demonstrations**

I. Introduction

Consider the task of setting a dinner table. It involves placing the appropriate serving utensils and cutlery according to the dishes being served. It might also require placing objects in a particular partial order either due to the fact that they are stacked on top of each other, or due to certain social conventions. Linear temporal logic (LTL) provides an expressive grammar for capturing these non-Markovian constraints. Incorporating LTL formulas as specifications for reinforcement learning ([1], [2], [3]) extends the possibility of applying reinforcement learning algorithms to complex non-Markovian tasks.

However, formalizing sound and complete specifications as an LTL formula is non-trivial. Thus it is desirable to infer specifications through demonstrations ([4], [5], [6]), or natural language instructions [7] provided by domain experts. Further some works also elicit specifications from multiple experts [8]. However these sources of specifications are inherently ambiguous or potentially contradictory. For example, while learning the dinner table-setting task from demonstrations, if the learner only observes the teacher place the dishes before the cutlery, this could be purely coincidental or it could be a social convention. An ideal learner would simultaneously consider both these hypotheses while performing the task. Or in case of eliciting preferences from multiple experts, two culinary experts might have strong but differing opinions about the same. Thus in a general setting, the task specifications cannot be stated as a single LTL formula, but as a belief over multiple LTL formulas ([4], [5]).

In this paper, we introduce a novel problem formulation for planning with uncertain specifications (PUnS), which allows task specifications to be expressed as a distribution over multiple LTL formulas. We identify four evaluation criteria that capture the semantics of satisfying a belief over LTL formulas and analyze the nature of the task executions they entail. Finally, we demonstrate that an instance of PUnS is equivalent to a reward machine ([9], [10]), therefore an equivalent MDP formulation exists for all instances of PUnS.

II. Related Work

Prior research into reinforcement learning has indicated great promise in sequential decision-making tasks, with breakthroughs in handling large-dimensional state spaces such as Atari games [11], continuous action spaces ([12], [13]), sparse rewards ([14], [15]), and all of these challenges in combination [16]. These were made possible due to the synergy between off-policy training methods and the expressive power of neural networks. This body of work has largely focused on algorithms for reinforcement learning rather than the source of task specifications; however, reward engineering is crucial to achieving high performance, and is particularly difficult in complex tasks where the user's intent can only be represented as a collection of preferences [8] or a belief over logical formulas inferred from demonstrations [4].

Reward design according to user intent has primarily been studied in the context of Markovian reward functions. Singh et al. [17] first defined the problem of optimal reward design with respect to a distribution of target environments. Ratner et al. [18] and Hadfield-Menell et al. [19] defined inverse reward design as the problem of inferring the true desiderata of a task from proxy reward functions provided by users for a set of task environments. Sadigh et al. [20] developed a model to utilize binary preferences over executions as a means of inferring the true reward. Regan and Boutillier [21] proposed algorithms for computation of robust policies that satisfy the minimax regret criterion. However, all of these works only allow for

Manuscript received: September, 10, 2019; Revised December, 30, 2019; Accepted February, 8, 2020 .

This paper was recommended for publication by Editor Dongheui Lee upon evaluation of the Associate Editor and Reviewers' comments.

[1]Ankit Shah is a PhD candidate at the Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. ajshah@mit.edu

[2]Shen Li is a PhD student Computer Science and Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. shenli@mit.edu

[3]Julie Shah is an associate professor at the Massachusetts Institute of Technology

Markovian reward functions; our proposed framework handles uncertain, non-Markovian specification expressed as a belief over LTL formulas.

LTL is an expressive language for representing non-Markovian properties. There has been considerable interest in enabling LTL formulas to be used as planning problem specifications, with applications in symbolic planning ([8],[22], [23]) and hybrid controller synthesis [24]. There has also been growing interest in the incorporation of LTL specifications into reinforcement learning. Aksaray et al. [1] proposed using temporal logic variants with quantitative semantics as the reward function. Littman et al. [2] compiled an LTL formula into a specification MDP with binary rewards and introduced geometric-LTL, a bounded time variant of LTL where the time horizon is sampled from a geometric distribution. Toro Icarte et al. [3] proposed LPOPL, an algorithm leveraging progressions and multi-task learning, to compute policies to satisfy any co-safe LTL [25] specification. Lacerda et al. [26] also developed planners that resulted in maximal completion of tasks for unsatisfiable specifications for co-safe LTL formulas. Within the domain of symbolic planning, expressing task objectives, constraints, and preferences using a set of LTL formulas was introduced with PDDL 3.0 [27] for the fifth International Planning Competition (IPC-5) [28]. Baier et al. [29] proposed a symbolic planner based capable of handling temporally extended preferences defined in PDDL 3.0, while the algorithm developed by Camacho et al. [23] is also capable of handling non-deterministic planning domains. However, while these works consider LTL specifications directly defined by the user, our framework considers the problem of planning with a belief over LTL formulas as the task specification.

Prior research into expressing non-Markov reward functions for planning under uncertainty has also explored the relationship between reward functions, formal languages and their finite state machine representations. Bacchus et al. [30] defined temporally extended reward functions (TERF) over a set of past-tense LTL formulas, and demonstrated the existence of an MDP equivalent to the non-Markov planning problem. In recent work, Camacho et al. [10] explored the relationship between formal languages and reward machines defined by Toro Icarte et al. [9]. They demonstrated that goal specifications written in multiple formal languages can be translated into equivalent reward machines, while the reverse transformation was not always possible. Further Camacho et al. [31] proposed a reward shaping to improve the convergence of reinforcement learning algorithms while planning for sparse rewards generated by a reward machine. We demonstrate that the MDP reformulation of an instance of the PUnS problem is an instance of a reward machine.

III. Preliminaries

A. Linear Temporal Logic

Linear temporal logic (LTL), introduced by Pnueli [32], provides an expressive grammar for describing temporal behaviors. An LTL formula is composed of atomic propositions (discrete time sequences of Boolean literals) and both logical and temporal operators, and is interpreted over traces $[\boldsymbol{\alpha}]$ of the set of propositions, $\boldsymbol{\alpha}$. The notation $[\boldsymbol{\alpha}],t \models \varphi$ indicates that φ holds at time t. The trace $[\boldsymbol{\alpha}]$ satisfies φ (denoted as $[\boldsymbol{\alpha}] \models \varphi$) iff $[\boldsymbol{\alpha}],0 \models \varphi$. The minimal syntax of LTL can be described as follows:

$$\varphi ::= p \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \mathbf{X}\varphi_1 \mid \varphi_1 \mathbf{U} \varphi_2 \tag{1}$$

p is an atomic proposition, and φ_1 and φ_2 represent valid LTL formulas. The operator $\mathbf{X}$ is read as "next" and $\mathbf{X}\varphi_1$ evaluates as true at time t if φ_1 evaluates to true at $t+1$. The operator $\mathbf{U}$ is read as "until" and the formula $\varphi_1\mathbf{U}\varphi_2$ evaluates as true at time t_1 if φ_2 evaluates as true at some time $t_2 > t_1$ and φ_1 evaluates as true for all time steps t, such that $t_1 \leq t \leq t_2$. We also use the additional propositional logic operators $\wedge$ (and) and $\mapsto$ (implies), as well as other higher-order temporal operators: $\mathbf{F}$ (eventually) and $\mathbf{G}$ (globally). $\mathbf{F}\varphi_1$ evaluates to true at t_1 if φ_1 evaluates as true for some $t \geq t_1$. $\mathbf{G}\varphi_1$ evaluates to true at t_1 if φ_1 evaluates as true for all $t \geq t_1$.

The "safe" and "co-safe" subsets of LTL formulas have been identified in prior research ([25], [33], [34]). A "co-safe" formula is one that can always be verified by a trace of a finite length, whereas a "safe" formula can always be falsified by a finite trace. Any formula produced by the following grammar is considered "co-safe":

$$\varphi_{co-safe} ::= \top \mid p \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \varphi_1 \mathbf{U} \varphi_2 \tag{2}$$

Similarly, any formula produced by the following grammar is considered "safe":

$$\varphi_{safe} ::= \bot \mid p \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \varphi_1 \mathbf{R} \varphi_2 \tag{3}$$

A formula expressed as $\varphi = \varphi_{safe} \wedge \varphi_{co-safe}$ belongs to the Obligation class of formulas presented in Manna and Pnueli's [34] temporal hierarchy.

Finally, a progression $\text{Prog}(\varphi, \alpha_t)$ over an LTL formula with respect to a truth assignment α_t at time t is defined such that for a trace of truth assignments over propositions $[\boldsymbol{\alpha}]$: $[\boldsymbol{\alpha}],t \models \varphi$ *iff* $[\boldsymbol{\alpha}],t+1 \models \text{Prog}(\varphi, \alpha_t)$, where α_t is the truth value of the propositions in the trace $[\boldsymbol{\alpha}]$ at time t. Thus, a progression of an LTL formula with respect to a truth assignment is a formula that must hold at the next time step in order for the original formula to hold at the current time step. Bacchus and Kabanza [35] defined a list of progression rules for the temporal operators in Equations 1, 2, and 3.

B. Belief over Specifications

In this paper, we define the specification of our planning problem as a belief over LTL formulas. A belief over LTL formulas is defined as a probability distribution with support over a finite set of formulas with the probability mass function $P : \boldsymbol{\varphi} \to [0,1]$; where $\boldsymbol{\varphi}$ is the set of LTL formulas belonging to the Obligation class defined by Manna and Pnueli [34]. The support of $P(\varphi)$ is restricted to a finite set of formulas $\{\varphi\}$. The distribution represents the probability of a particular formula being the true specification.

C. Model-free Reinforcement Learning

A Markov decision process (MDP) is a planning problem formulation defined by the tuple $\mathcal{M} = \langle S, A, T, R \rangle$, where S is the set of all possible states, A is the set of all possible actions, and $T := P(s' \mid s, a)$ is the probability distribution that the next state will be $s' \in S$ given that the current state is $s \in S$ and the action taken at the current time step is $a \in A$. $R : S \to \mathbb{R}$ represents the reward function that returns a scalar value given a state. Watkins and Dayan proposed Q-learning [36], an off-policy, model-free algorithm to compute optimal policies in discrete MDPs. The Q-value function $Q_\pi(s,a)$ is the expected discounted value under a policy $\pi(a \mid s)$. In a model-free setting, the transition function is not known to the learner, and the Q-value is updated by the learner acting within the environment and observing the resulting reward. If the Q-value is updated while not following the current estimate of the optimal policy, it is considered "off-policy" learning. Given an initial estimate of the Q-value $Q(s,a)$, the agent performs an action a from state s to reach s' while collecting a reward r and a discounting factor $\gamma \in [0,1)$. The Q-value function is then updated as follows:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in A} Q(s',a')) \quad (4)$$

IV. Planning with Uncertain Specifications (PUnS)

The problem of planning with uncertain specifications (PUnS) is formally defined as follows: The state representation of the learning and task environment is denoted by $x \in \mathcal{X}$, where $\mathcal{X}$ is a set of features that describe the physical state of the system. The agent has a set of available actions, $\boldsymbol{A}$. The state of the system maps to a set of finite known Boolean propositions, $\boldsymbol{\alpha} \in \{0,1\}^{n_{prop}}$, through a known labeling function, $f : \mathcal{X} \to \{0,1\}^{n_{props}}$. The specification is provided as a belief over LTL formulas, $P(\varphi)$; $\varphi \in \{\varphi\}$, with a finite set of formulas in its support. The expected output of the planning problem is a stochastic policy, $\pi_{\{\varphi\}} : \mathcal{X} \times \boldsymbol{A} \to [0,1]$, that satisfies the specification.

The binary semantics of satisfying a single logical formula are well defined; however, there is no single definition for satisfying a belief over logical formulas. In this work, we present four criteria for satisfying a specification expressed as a belief over LTL, and express them as non-Markovian reward functions. A solution to PUnS optimizes the reward function representing the selected criteria. Next, using an approach inspired by LTL-to-automata compilation methods ([37]), we demonstrate the existence of an MDP that is equivalent to PUnS. The reformulation as an MDP allows us to utilize any reinforcement learning algorithm that accepts an instance of an MDP to solve the corresponding instance of PUnS.

A. Satisfying beliefs over specifications

A single LTL formula can be satisfied, dissatisfied, or undecided; however, satisfaction semantics over a distribution of LTL formulas do not have a unique interpretation. We identify the following four evaluation criteria, which capture the semantics of satisfying a distribution over specifications, and formulate each as a non-Markovian reward function:

1) **Most likely**: This criteria entails executions that satisfy the formula with the largest probability as per $P(\varphi)$. As a reward, this is represented as follows:

$$\begin{gathered} J([\boldsymbol{\alpha}]; P(\varphi)) = \mathbb{1}([\boldsymbol{\alpha}] \models \varphi^*) \\ \text{where } \varphi^* = \underset{\varphi \in \{\varphi\}}{\arg\max}\, P(\varphi) \end{gathered} \quad (5)$$

where

$$\mathbb{1}([\boldsymbol{\alpha}] \models \varphi) = \begin{cases} 1, & \text{if } [\boldsymbol{\alpha}] \models \varphi \\ -1, & \text{otherwise} \end{cases} \quad (6)$$

2) **Maximum coverage:** This criteria entails executions that satisfy the maximum number of formulas in support of the distribution $P(\varphi)$. As a reward function, it is represented as follows:

$$J([\boldsymbol{\alpha}]; P(\varphi)) = \sum_{\varphi \in \{\varphi\}} \mathbb{1}([\boldsymbol{\alpha}] \models \varphi) \quad (7)$$

3) **Minimum regret:** This criteria entails executions that maximize the hypothesis-averaged satisfaction of the formulas in support of $P(\varphi)$. As a reward function, this is represented as follows:

$$J([\boldsymbol{\alpha}]; P(\varphi)) = \sum_{\varphi \in \{\varphi\}} P(\varphi)\mathbb{1}([\boldsymbol{\alpha}] \models \varphi) \quad (8)$$

4) **Chance constrained:** Suppose the maximum probability of failure is set to δ, with $\boldsymbol{\varphi}^\delta$ defined as the set of formulas such that $\sum_{\varphi \in \boldsymbol{\varphi}^\delta} P(\varphi) \geq 1 - \delta$; and $P(\varphi') \leq P(\varphi)\ \forall\ \varphi' \notin \boldsymbol{\varphi}^\delta, \varphi \in \boldsymbol{\varphi}^\delta$. This is equivalent to selecting the most-likely formulas until the cumulative probability density exceeds the risk threshold. As a reward, this is represented as follows:

$$J([\boldsymbol{\alpha}]; P(\varphi)) = \sum_{\varphi \in \boldsymbol{\varphi}^\delta} P(\varphi)\mathbb{1}([\boldsymbol{\alpha}] \models \varphi) \quad (9)$$

Each of these four criteria represents a "reasonable" interpretation of satisfying a belief over LTL formulas, with the choice between the criteria dependent upon the relevant application. In a preference elicitation approach proposed by Kim et al. [8], the specifications within the set $\{\varphi\}$ are provided by different experts. In such scenarios, it is desirable to satisfy the largest common set of specifications, making *maximum coverage* the most suitable criteria. When the specifications are inferred from task demonstrations (such as in the case of Bayesian specification inference [4]), *minimum regret* would be the natural formulation. However, if the formula distribution is skewed towards a few likely formulas with a long tail of low-probability formulas, the *chance constrained* or *most likely* criteria can be used to reduce computational overhead in resource-constrained or time-critical applications.

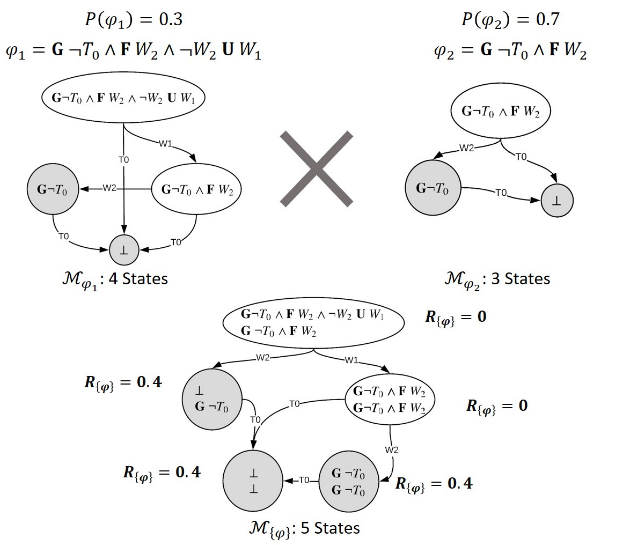

$P(\varphi_1) = 0.3$
$\varphi_1 = \mathbf{G}\,\neg T_0 \wedge \mathbf{F}\, W_2 \wedge \neg W_2\, \mathbf{U}\, W_1$
$P(\varphi_2) = 0.7$
$\varphi_2 = \mathbf{G}\,\neg T_0 \wedge \mathbf{F}\, W_2$
$\mathbf{G}\neg T_0 \wedge \mathbf{F}\, W_2 \wedge \neg W_2\, \mathbf{U}\, W_1$
W1
T0
W2
$\mathbf{G}\neg T_0$
$\mathbf{G}\neg T_0 \wedge \mathbf{F}\, W_2$
$\perp$
$\mathcal{M}_{\varphi_1}$: 4 States
$\mathcal{M}_{\varphi_2}$: 3 States
$R_{\{\varphi\}} = 0$
$R_{\{\varphi\}} = 0.4$
$\mathcal{M}_{\{\varphi\}}$: 5 States

Fig. 1: Example compilation process with $\{\varphi\} = \{\varphi_1, \varphi_2\}$ and the *minimum regret* criterion. The deterministic MDPs $\mathcal{M}_{\varphi_1}$, and $\mathcal{M}_{\varphi_2}$ are composed through a cross product to yield the deterministic MDP $\mathcal{M}_{\{\varphi\}}$ corresponding to the set $\{\varphi\}$. The states of $\mathcal{M}_{\{\varphi\}}$ producing a non-zero reward are shaded in gray. Note that while a naïve enumeration would yield a discrete state space with 12 states, the breadth-first enumeration generates a minimal set with five states. For clarity, only the edges corresponding to change of truth value of only a single proposition are shown, while self transitions are not shown.

B. Specification-MDP compilation

We demonstrate that an equivalent MDP exists for all instances of PUnS. We represent the task environment as an MDP sans the reward function, then compile the specification $P(\varphi)$ into an automaton with terminal reward generating states. The MDP equivalent of the PUnS problem is generated through the cross-product of the environment MDP with the automaton representing $P(\varphi)$. Figure 1 depicts the compilation of an illustrative PUnS problem into an automaton representing a deterministic MDP. The belief over formulas, $P(\varphi)$ has support $\{\varphi\} = \{\varphi_1 := \mathbf{G}\neg T_0 \wedge \mathbf{F}\ W_2 \wedge \neg W_2\ \mathbf{U}\ W1,\ \varphi_2 := \mathbf{G}\neg T_0 \wedge \mathbf{F} W_2\}$; $P(\varphi_1) = 0.3$, and $P(\varphi_2) = 0.7$. φ_1 is satisfied if "T_0" never becomes true and "W_1" and "W_2" become true in that order. φ_2 is satisfied if "T_0" never becomes true and "W_2" becomes true eventually.

Given a single LTL formula, φ, a Büchi automaton can be constructed which accepts traces that satisfy the property represented by φ [33]. An algorithm to construct the automaton was proposed by Gerth et al. [37]. The automata are directed graphs where each node represents a LTL formula φ' that the trace must satisfy from that point onward in order to be accepted by the automaton. An edge, labeled by the truth assignment at a given time α_t, connects a node to its progression, $\mathrm{Prog}(\varphi', \alpha_t)$. Our decision to restrict φ to the Obligation class of temporal properties ($\varphi_{safe} \wedge \varphi_{co-safe}$) ensures that the automaton constructed from φ is deterministic and will have terminal states that represent $\top$, $\perp$, or φ_{safe} [34]. When planning with a single formula, these terminal states are the reward-generating states for the overall MDP, as seen in approaches proposed by Littman et al. [2] and Toro Icarte et al. [3].

An LTL formula can be represented by an equivalent deterministic MDP described by the tuple $\mathcal{M}_\varphi = \langle \{\varphi'\}, \{0,1\}^{n_{prop}}, T, R\rangle$, with the states representing the possible progressions of φ and the actions representing the truth assignments causing the progressions ([2], [3]). The transition function is defined as follows:

$$T_\varphi(\varphi'_1, \varphi'_2, \alpha) = \begin{cases} 1, & \text{if } \varphi'_2 = \mathrm{Prog}(\varphi'_1, \alpha) \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

The reward function R is a function of the MDP state, and defined as follows:

$$R_\varphi(\varphi') = \begin{cases} 1, & \text{if } \varphi' = \top \text{ or } \varphi' = \varphi_{safe} \\ -1, & \text{if } \varphi' = \perp \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

The equivalent MDPs $\mathcal{M}_{\varphi_1}$, and $\mathcal{M}_{\varphi_2}$ corresponding to φ_1 and φ_2, with four and three states respectively, are depicted in Figure 1. Each state encodes the temporal property that must hold in the future once the MDP enters that state. For example, $\mathcal{M}_{\varphi_1}$ is initially in the state labeled as $\mathbf{G}\neg T_0 \wedge \mathbf{F} W_2 \wedge \neg W_2\ \mathbf{U}\ W_1$. Once the proposition "$W_1$" evaluates as true, the MDP enters the state labeled by $\mathbf{G}\neg T_0 \wedge \mathbf{F} W_2$, that encodes the temporal property that in the future, "W_2" must eventually evaluate as true. Note that in Figure 1, only the edges corresponding to changing truth values of a single propositions are depicted for clarity.

For an instance of PUnS with specification $P(\varphi)$ and support $\{\varphi\}$, a deterministic MDP is constructed by computing the cross-product of MDPs of the component formulas. Let $\langle \boldsymbol{\varphi'} \rangle = \langle \varphi'^1, \ldots \varphi'^n \rangle$; $\forall \varphi'^i \in \{\varphi\}$ be the progression state for each of the formulas in $\{\varphi\}$; the MDP equivalent of $\{\varphi\}$ is then defined as $\mathcal{M}_{\{\varphi\}} = \langle \{\langle \boldsymbol{\varphi'} \rangle\}, \{0,1\}^{n_{prop}}, T_{\{\varphi\}}, R_{\{\varphi\}} \rangle$. Here, the states are all possible combinations of the component formulas' progression states, and the actions are propositions' truth assignments. The transition is defined as follows:

$$T_{\{\varphi\}}(\langle \boldsymbol{\varphi'_1} \rangle, \langle \boldsymbol{\varphi'_2} \rangle, \alpha) = \begin{cases} 1, & \text{if } \varphi'^i_2 = \mathrm{Prog}(\varphi'^i_1, \alpha) \forall i \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

This MDP reaches a terminal state when all of the formulas comprising $\{\varphi\}$ have progressed to their own terminal states. The reward is computed using one of the criteria represented by Equations 5, 7, 8, or 9, with $\mathbb{1}(\ldots)$ replaced by $R_\varphi(\varphi')$. Note that while $\mathbb{1}(\ldots)$ has two possible values (1 when the formula is satisfied and -1 when it is not) $R_\varphi(\varphi')$ has three possible values (1 when φ has progressed to $\top$ or φ_{safe}, -1 when φ has progressed to $\perp$, or 0 when φ has not progressed to a terminal state). Thus, the reward is non-zero only in a terminal state.

Consider the example PUnS problem depicted in Figure 1. The initial state of $\mathcal{M}_{\{\varphi\}}$ is labeled $\langle \mathbf{G}\neg T_0 \wedge \mathbf{F} W_2 \wedge \neg W_2\ \mathbf{U}\ W_1, \mathbf{G}\neg T_0 \wedge \mathbf{F} W_2 \rangle$. From this state, if W_2 evaluates as true, the MDP transitions into a state labeled $\langle \perp, \mathbf{G}\neg T_0 \rangle$, where

φ_1 is dissatisfied, and φ_2 has progressed to φ_{safe}. $\mathcal{M}_{\{\varphi\}}$ has three terminal states labeled $\langle \bot, \mathbf{G}\neg T_0\rangle$, $\langle \mathbf{G}\neg T_0,\ \mathbf{G}\neg T_0\rangle$ and $\langle \bot, \bot\rangle$ with corresponding rewards of 0.4, 1.0, and -1.0 as per the *minimum regret* criterion.

In the worst case, the size of the automaton of $\{\varphi\}$ is exponential in $|\{\varphi\}|$. In practice, however, many formulas contained within the posterior may be logically correlated. In the example depicted in Figure 1, a naïve enumeration of the states would have resulted in 12 discrete states. However there are certain states such as $\langle \mathbf{G}\neg T_0, \bot\rangle$, corresponding to φ_1 being satisfied and φ_2 being dissatisfied that are impossible. In fact, the minimal state space only has five reachable states as depicted in Figure 1. To compute a minimal reachable set of states, we start from $\langle \boldsymbol{\varphi}\rangle$ and perform a breadth-first enumeration. As the deterministic MDP $\mathcal{M}_{\{\varphi\}}$ has a finite number of states, and an output function $R_{\{\varphi\}}$ dependent only on the current state, it is an instance of a reward machine [9], [10].

We represent the task environment as an MDP without a reward function using the tuple $\mathcal{M}_{\mathcal{X}} = \langle \mathcal{X}, \boldsymbol{A}, T_{\mathcal{X}}\rangle$. The cross product of $\mathcal{M}_{\mathcal{X}}$ and $\mathcal{M}_{\{\varphi\}}$ results in an MDP: $\mathcal{M}_{Spec} = \langle \{\langle \boldsymbol{\varphi'}\rangle\} \times \mathcal{X}, \boldsymbol{A}, T_{Spec}, R_{\{\varphi\}}\rangle$. The transition function of $\mathcal{M}_{\{\varphi\}}$ is defined as follows:

$$\begin{aligned} T_{Spec}(\langle\langle \boldsymbol{\varphi'_1}\rangle, x_1\rangle, \langle\langle \boldsymbol{\varphi'_2}\rangle, x_2\rangle, a) = \\ T_{\{\varphi\}}(\langle \boldsymbol{\varphi'_1}\rangle, \langle \boldsymbol{\varphi'_2}\rangle, f(x_2)) \times T_{\mathcal{X}}(x_1, x_2, a) \end{aligned} \tag{13}$$

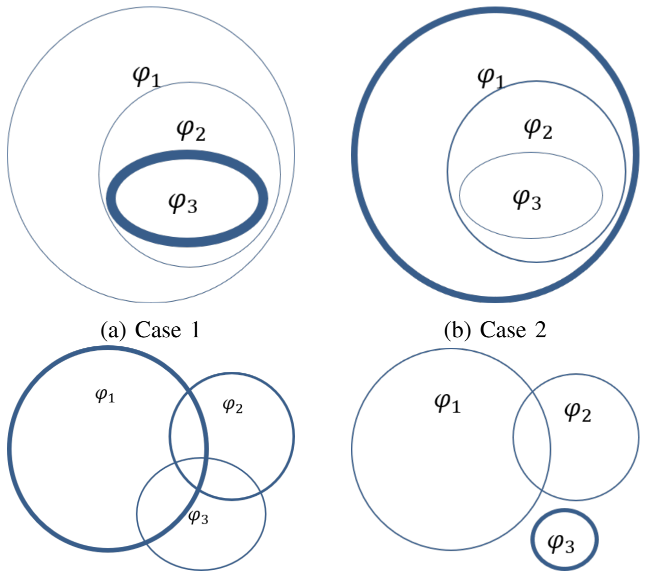
$\mathcal{M}_{Spec}$ is an equivalent reformulation of PUnS as an MDP, creating the possibility of leveraging recent advances in reinforcement learning for PUnS. In Section V, we demonstrate examples of PUnS trained using off-policy reinforcement learning algorithms.

C. *Counterfactual updates in a model-free setting*

Toro Icarte et al. ([9], [3]) demonstrated that reward machines allow for off-policy updates for each state in the reward machine. Constructing $\mathcal{M}_{Spec}$ as a composition of $\mathcal{M}_{\mathcal{X}}$ and $\mathcal{M}_{\{\varphi\}}$ results in the following properties: the reward function is only dependent upon $\langle \boldsymbol{\varphi}\rangle$, the state of $\mathcal{M}_{\{\varphi\}}$; the action availability only depends upon x, the state of $\mathcal{M}_{\mathcal{X}}$; and the stochasticity of transitions is only in $T_{\mathcal{X}}$, as $T_{\{\varphi\}}$ is deterministic. These properties allow us to exploit the underlying structure of $\mathcal{M}_{Spec}$ in a model-free learning setting. Let an action $a \in \boldsymbol{A}$ from state $x_1 \in \mathcal{X}$ result in a state $x_2 \in \mathcal{X}$. As $T_{\{\varphi\}}$ is deterministic, we can use this action update to apply a Q-function update (Equation 4) to all states described by $\langle\langle \boldsymbol{\varphi'}\rangle, x_1\rangle\ \forall\ \langle \boldsymbol{\varphi'}\rangle \in \{\langle \boldsymbol{\varphi}\rangle\}$.

V. Evaluations

In this section, we first explore how the choice of criteria represented by Equations 5, 7, 8, and 9 results in qualitatively different performance by trained RL agents. Then, we demonstrate how the MDP compilation can serve to train an agent on a real-world task involving setting a dinner table with specifications inferred from human demonstrations, as per Shah et al. [4]. We also demonstrate the value of counterfactual Q-value updates for speeding up the agent's learning curve.



(c) Case 3 (d) Case 4

Fig. 2: Comparisons between different types of distributions over specifications. In each case , the size of the set is proportional to the number of executions satisfying the specification, and the thickness of the boundary is proportional to the probability mass assigned to that specification.

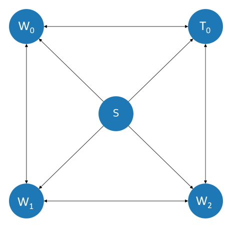
A. *Synthetic Examples*

The choice of the evaluation criterion impacts the executions it entails based on the nature of the distribution $P(\varphi)$. Figure 2 depicts examples of different distribution types. Each figure is a Venn diagram where each formula φ_i represents a set of executions that satisfy φ_i. The size of the set represents the number of execution traces that satisfy the given formula, while the thickness of the set boundary represents its probability. Consider the simple discrete environment depicted in Figure 3a: there are five states, with the start state in the center labeled 'S' and the four corner states labeled "T_0", "W_0", "W_1", and "W_2". The agent can act to reach one of the four corner states from any other state, and that action is labeled according to the node it is attempting to reach.

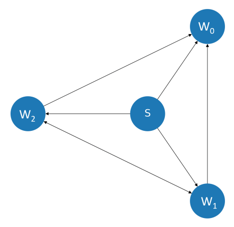
Case 1: Figure 2a represents a distribution where the most restrictive formula of the three is also the most probable. All criteria will result in the agent attempting to perform executions that adhere to the most restrictive specification.

Case 2: Figure 2b represents a distribution where the most likely formula is the least restrictive. The *minimum regret* and *maximum coverage* rewards will result in the agent producing executions that satisfy φ_3, the most restrictive formula; however, using the *most likely* criteria will only generate executions that satisfy φ_1. With the chance-constrained policy, the agent begins by satisfying φ_3 and relaxes the satisfactions as risk tolerance is decreased, eventually satisfying φ_1 but not necessarily φ_2 or φ_3.

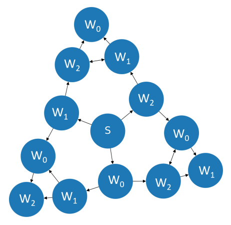
Case 3: Case 3 represents three specifications that share a common subset but also have subsets that satisfy neither of the other specifications. Let the scenario specification be $\{\varphi\} = \{\mathbf{G}\neg T_0 \wedge \mathbf{F}W_0, \mathbf{G}\neg T_0 \wedge \mathbf{F}W_1, \mathbf{G}\neg T_0 \wedge \mathbf{F}W_2\}$ with assigned



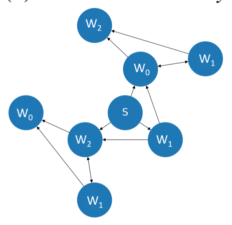
(a) Task MDP



(b) Case 3: *most likely*

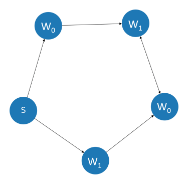


(c) Case 3: *min regret*

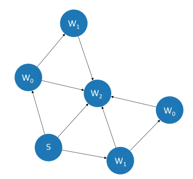


(d) Case 3: *chance constrained*

Fig. 3: Figure 3a depicts the transition diagram for the example MDP. Figures 3b, 3c, and 4a depict the exploration graph of agents trained with different evaluation criteria for distributions with an intersecting set of satisfying executions.



(a) Case 4: *max coverage*



(b) Case 4: *min regret*

Fig. 4: Figures 4a and 4b depict the exploration graph of agents trained with different evaluation criteria for distributions without an intersecting set of satisfying executions.

probabilities to each of $0.4, 0.25,$ and 0.35, respectively. These specifications correspond to always avoiding "T_0" and visiting either "W_0", "W_1", or "W_2". For each figure of merit defined in Section IV-A, the Q-value function was estimated using $\gamma = 0.95$ and an ε-greedy exploration policy. A softmax policy with temperature parameter 0.02 was used to train the agent, and the resultant exploration graph of the agent was recorded. The *most likely* criterion requires only the first formula in $\{\varphi\}$ to be satisfied; thus, the agent will necessarily visit "W_0" but may or may not visit "W_1" or "W_2", as depicted in Figure 3b. With either *maximum coverage* or *minimum regret* serving as the reward function, the agent tries to complete executions that satisfy all three specifications simultaneously. Therefore, each task execution ends with the agent visiting all three nodes in all possible orders, as depicted in Figure 3c. Finally, in the chance-constrained setting with risk level $\delta = 0.3$, the automaton compiler drops the second specification; the resulting task executions always visit "W_0" and "W_2" but not necessarily "W_1", as depicted in Figure 3d.

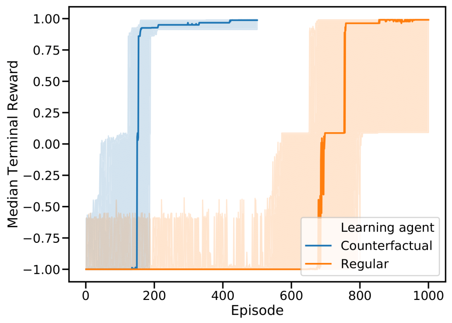
Case 4: Case 4 depicts a distribution where an intersecting subset does not exist. Let the scenario specifications be $\{\varphi\}$ $= \{\mathbf{G}\neg T0 \wedge \mathbf{G}\neg W_2 \wedge \mathbf{F}W_1, \mathbf{G}\neg T_0 \wedge \mathbf{G}\neg W_2 \wedge \mathbf{F}W_1, \mathbf{G}\neg T_0 \wedge \mathbf{F}W_2\}$, with probabilities assigned to each of $0.05, 0.15,$ and 0.8, respectively. The first two formulas correspond to the agent visiting either "W_1" or "W_0" but not "W_2". The third specification is satisfied when the agent visits "W_2"; thus, any execution that satisfies the third formula will not satisfy the first two. The first two formulas also have an intersecting set of satisfying executions when both "W_0" and "W_1" are visited, corresponding to Case 4 from Figure 2d. Optimizing for *max coverage* will result in the agent satisfying both the first and the second formula but ignoring the third, as depicted in Figure 4a. However, when using the *minimum regret* formulation, the probability of the third specification is higher than the combined probability of the first two formulas; thus, a policy learned to optimize *minimum regret* will ignore the first two formulas and always end an episode by visiting "$W2$", as depicted in Figure 4b. The specific examples and exploration graphs for the agents in each of the scenarios in Figure 2 and for each reward formulation in Section IV-A are provided in the supplemental materials.

B. Planning with Learned Specifications: Dinner Table Domain

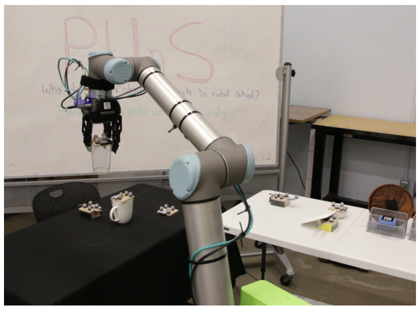
We also formulated the task of setting a dinner table as an instance of PUnS, using the dataset and resulting posterior distributions provided by Shah et al. [4]. This task features eight dining set pieces that must be organized in a configuration depicted in Figure 5a. In order to successfully complete the task, the agent must place each of the eight objects in the final configuration. As the dinner plate, small plate and the bowl were stacked, they had to be placed in that particular partial order. The propositions $\boldsymbol{\alpha}$ comprise eight Boolean variables associated with whether an object is placed in its correct position. The original dataset included 71 demonstrations; Bayesian specification inference was used to compute the posterior distributions over LTL formulas for different training set sizes.

For the purpose of planning, the task environment MDP $\mathcal{M}_{\mathcal{X}}$ was simulated. Its state was defined by the truth values of the eight propositions defined above; thus, it had 256 unique states. The action space of the robot was the choice of which object to place next. Once an action was selected, it had an 80% chance of success as per the simulated transitions. For this scenario, we selected the posterior distribution trained with 30 training demonstrations, as it had the largest uncertainty in true specification. This distribution $P(\varphi)$ had 25 unique formulas in its support $\{\varphi\}$. As per the expected value of the intersection over union metric, the belief was 85% similar to the true specification. The true specification itself was part of the support, but was only the fourth most likely formula, as per the distribution. The deterministic MDP $\mathcal{M}_{\{\varphi\}}$ compiled from $P(\varphi)$ had 3,025 distinct states; thus, the cross-product of $\mathcal{M}_{\{\varphi\}}$ and $\mathcal{M}_{\mathcal{X}}$ yielded $\mathcal{M}_{Spec}$ with 774,400 unique states and the same action space as $\mathcal{M}_{\mathcal{X}}$. We chose the *minimum regret* criteria to construct the reward function, and trained two learning agents using Q-learning with an ε-greedy policy ($\varepsilon = 0.3$): one with and one without off-policy updates. We

(a) Desired final configuration



(b) Learning Curves



(c) Task setup

Fig. 5: Figure 5a depicts the desired final configuration of objects. Figure 5b depicts the median terminal rewards and the 25th and 75th quartiles. Figure 5c presents the UR-10 arm performing the table-setting task.

Reward Type	Formulas included	$\mathcal{M}_{\{\varphi\}}$ States	True valid orders	Successful executions (On robot)	Unique orders (On robot)	Constraint violations (On robot)	Successful executions (Simulation)	Unique orders (Simulations)	Constraint violations (Simulations)
Min Regret	25	3025	6720	20	20	0	19997	1962	3
Most Likely	1	193	6720	12	20	8	9920	10215	10080
Chance Constrained ($\delta = 0.1$)	4	449	6720	20	20	0	19995	4253	5
Chance Constrained ($\delta = 0.3$)	3	353	6720	20	20	0	19987	4882	13

TABLE I: A summary of the experiments on the physical robot and simulated executions for four different reward compilations. The differences in the number of unique orderings recorded and the constraint violations demonstrates the risk-creativity trade-off inherent to PUnS.

evaluated the agent at the end of every training episode using an agent initialized with softmax policy (the temperature parameter was set to 0.01). The agent was allowed to execute 50 test episodes, and the terminal value of the reward function was recorded for each; this was replicated 10 times for each agent. All evaluations were conducted on a desktop with i7-7700K and 16 GB of RAM. Our code is included in the supplementary materials, and is adapted from LPOPL [9][1].

The statistics of the learning curve are depicted in Figure 5b. The solid line represents the median value of terminal reward across evaluations collected from all training runs. The error bounds indicate the 75th and 25th percentile. The maximum value of the terminal reward is 1 when all formulas in the support $\{\varphi\}$ are satisfied, and the minimum value is -1 when all formulas are not satisfied. The learning curves indicate that the agent that performed Q-value updates for all states of $\mathcal{M}_{\{\varphi\}}$ learned faster and had less variability in its task performance across training runs compared with the one that did not perform counterfactual updates. This provides additional empirical evidence to suggest that off-policy updates to each reward machine state improve the sample complexity as observed by Toro Icarte et al. ([3], [9]).

We implemented the learned policy with predesigned motion primitives on a UR-10 robotic arm. We observed during evaluation runs that the robot never attempted to violate any temporal ordering constraint. The stochastic policy also made it robust to some environment disturbances: for example, if one of the objects was occluded, the robot finished placing the other objects before waiting for the occluded object to become visible again[2].

Next to examine the trade-off between creativity of performing the task and risk aversion, we repeated the training and testing with the $\mathcal{M}_{\{\varphi\}}$ compiled with the *most likely* and the *chance constrained* criteria with $\delta = \{0.1, 0.3\}$. For each of the trained agents, we recorderd 20 physical executions by deploying the policy on the robot and we also ran 20000 simulated test episodes for each instance of the PUnS MDP $\mathcal{M}_{Spec}$. During the simulated and physical test runs, we recorded the number of unique placement sequences and the number of specification violations. The results are tabulated in Table I.

Assuming that the dinner plate, the small plate and the bowl must be placed in that partial order, there are 6720 unique valid orderings for placing the eight objects. The policies trained as per *min regret* and both the *chance constrained* criteria generated 20 unique orderings in the physical test executions. The simulated tests reveal that the policy trained in accordance with *minimum regret* criterion executes the task with fewer unique orders than the policy trained with both the *chance constrained* criteria. Both the physical executions and the simulations reveal that the policy trained with the *most likely* criterion performs the task with many constraint violations because the most likely formula does not include an ordering constraint existing in the ground truth specification. This demonstrates a three-way tradeoff between computational complexity in terms of the additional states to consider while planning, the creativity displayed by the policy in terms of the unique executions discovered and the risk of specification violation. The *min regret* policy is the most risk averse but

[1]https://bitbucket.org/RToroIcarte/lpopl

[2]example executions can be viewed at https://youtu.be/LrIh_jbnfmo

also the least creative while the *chance constrained* policies demonstrate a higher creativity but with more constraint violations.

VI. Conclusions

In this work, we formally define the problem of planning with uncertain specifications (PUnS), where the task specification is provided as a belief over LTL formulas. We propose four evaluation criteria that define what it means to satisfy a belief over logical formulas, and discuss the type of task executions that arise from the various choices. We also present a methodology for compiling PUnS as an equivalent MDP using LTL compilation tools adapted to multiple formulas. We also demonstrate that MDP reformulation of PUnS can be solved using off-policy algorithms with counterfactual updates for a synthetic example and a real-world task. Although we restricted the scope of this paper to discrete task environment MDPs, this technique is extensible to continuous state and action spaces; we plan to explore this possibility in future work.

References

[1] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *2016 IEEE 55th Conference on Decision and Control*, pp. 6565–6570, IEEE, 2016.

[2] M. L. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan, "Environment-independent task specifications via GLTL," *arXiv preprint arXiv:1704.04341*, 2017.

[3] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Teaching multiple tasks to an RL agent using LTL," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 452–461, International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[4] A. Shah, P. Kamath, J. A. Shah, and S. Li, "Bayesian inference of temporal task specifications from demonstrations," in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 3804–3813, Curran Associates, Inc., 2018.

[5] J. Kim, C. Muise, A. Shah, S. Agarwal, and J. Shah, "Bayesian inference of linear temporal logic specifications for contrastive explanations," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 5591–5598, International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[6] A. Camacho and S. A. McIlraith, "Learning interpretable models in linear temporal logic," in *Proceedings of the Twenty-Nineth International Conference on Automated Planning and Scheduling*, pp. 621–630, 2019.

[7] Y. Oh, R. Patel, T. Nguyen, B. Huang, E. Pavlick, and S. Tellex, "Planning with state abstractions for non-Markovian task specifications," in *Robotics: Science and Systems*, June 2019.

[8] J. Kim, C. Banks, and J. Shah, "Collaborative planning with encoding of users' high-level strategies," in *AAAI Conference on Artificial Intelligence*, 2017.

[9] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, pp. 2112–2121, 2018.

[10] 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 *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 6065–6073, 2019.

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

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

[13] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[14] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: a new approach for hard-exploration problems," *arXiv preprint arXiv:1901.10995*, 2019.

[15] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[16] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[17] S. Singh, R. L. Lewis, and A. G. Barto, "Where do rewards come from," in *Proceedings of the annual conference of the cognitive science society*, pp. 2601–2606, Cognitive Science Society, 2009.

[18] E. Ratner, D. Hadfield-Mennell, and A. Dragan, "Simplifying reward design through divide-and-conquer," in *Robotics: Science and Systems*, 2018.

[19] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan, "Inverse reward design," in *Advances in neural information processing systems*, pp. 6765–6774, 2017.

[20] D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia, "Active preference-based learning of reward functions," in *Robotics: Science and Systems*, 2017.

[21] K. Regan and C. Boutilier, "Robust policy computation in reward-uncertain mdps using nondominated policies," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[22] A. Camacho, J. A. Baier, C. Muise, and S. A. McIlraith, "Finite LTL synthesis as planning," in *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

[23] A. Camacho and S. A. McIlraith, "Strong fully observable non-deterministic planning with LTL and LTL-f goals," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 5523–5531, 2019.

[24] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

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

[26] B. Lacerda, D. Parker, and N. Hawes, "Optimal policy generation for partially satisfiable co-safe LTL specifications," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[27] A. Gerevini and D. Long, "Plan constraints and preferences in PDDL3," tech. rep., Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, 2005.

[28] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, "Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners," *Artificial Intelligence*, vol. 173, no. 5, pp. 619 – 668, 2009. Advances in Automated Plan Generation.

[29] J. A. Baier, F. Bacchus, and S. A. McIlraith, "A heuristic search approach to planning with temporally extended preferences," *Artificial Intelligence*, vol. 173, no. 5-6, pp. 593–618, 2009.

[30] F. Bacchus, C. Boutilier, and A. Grove, "Rewarding behaviors," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 1160–1167, 1996.

[31] A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith, "Non-Markovian rewards expressed in LTL: Guiding search via reward shaping (extended version)," in *First Workshop on Goal Specifications for Reinforcement Learning, collocated with ICML/IJCAI/AAMAS*, 2018.

[32] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pp. 46–57, IEEE, 1977.

[33] M. Y. Vardi, "An automata-theoretic approach to linear temporal logic," in *Logics for concurrency*, pp. 238–266, Springer, 1996.

[34] Z. Manna and A. Pnueli, *A hierarchy of temporal properties*. Department of Computer Science, 1987.

[35] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning," *Artificial intelligence*, vol. 116, no. 1-2, pp. 123–191, 2000.

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

[37] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *International Conference on Protocol Specification, Testing and Verification*, pp. 3–18, Springer, 1995.