Interpretable Apprenticeship Learning with Temporal Logic Specifications

Daniel Kasenberg and Matthias Scheutz

***Abstract*— Recent work has addressed using formulas in linear temporal logic (LTL) as specifications for agents planning in Markov Decision Processes (MDPs). We consider the inverse problem: inferring an LTL specification from demonstrated behavior trajectories in MDPs. We formulate this as a multiobjective optimization problem, and describe state-based ("what actually happened") and action-based ("what the agent expected to happen") objective functions based on a notion of "violation cost". We demonstrate the efficacy of the approach by employing genetic programming to solve this problem in two simple domains.**

I. Introduction

Apprenticeship learning, or learning behavior by observing expert demonstrations, allows artificial agents to learn to perform tasks without requiring the system designer to explicitly specify reward functions or objectives in advance. Apprenticeship learning has been accomplished in agents in stochastic domains, such as Markov Decision Processes (MDPs), by means of *inverse reinforcement learning* (IRL), in which agents infer some reward function presumed to underlie the observed behavior. IRL has recently been criticized, especially in learning ethical behavior [2], because the resulting reward functions (1) may not be easily explained, and (2) cannot represent complex temporal objectives.

Recent work (e.g., [5], [7], [21]) has proposed using linear temporal logic (LTL) as a specification language for agents in MDPs. An agent in a stochastic domain may be provided a formula in LTL, which it must satisfy with maximal probability. These approaches require the LTL specification to be specified a priori (e.g., by the system designer, although [6] construct specifications from natural language instruction).

This paper proposes combining the virtues of these approaches by inferring LTL formulas from observed behavior trajectories. Specifically, this inference problem can be formulated as multiobjective optimization over the space of LTL formulas. The two objective functions represent (1) the extent to which the given formula explains the observed behavior, and (2) the complexity of the given formula. The resulting specifications are interpretable, and can be subsequently applied to new problems, but do not need to be specified in advance by the system designer.

The key contributions of this work are (1) the introduction of this problem and its formulation as an optimization problem; and (2) the notion of violation cost, and the state- and action-based objectives based on this notion.

Authors are with the Department of Computer Science, Tufts University, Medford, MA 02155, USA. The corresponding author is dmk@cs.tufts.edu

In the remainder of the paper, we first discuss related work; we then describe our formulation of this problem as multiobjective optimization, defining a notion of "violation cost" and then describing state-based and action-based objectives, corresponding to inferring a specification from "what actually happened" and "what the demonstrator expected to happen" respectively. We demonstrate the usefulness of the formulation by using genetic programming to optimize these objectives in two domains, called SlimChance and CleaningWorld. We discuss issues pertaining to our approach and directions for future work, and summarize our results.

II. Related Work

The proposed problem draws primarily upon ideas from apprenticeship learning (particularly, inverse reinforcement learning), stochastic planning with temporal logic specifications, and inferring temporal logic descriptions of systems.

A. Apprenticeship Learning

Apprenticeship learning, the problem of learning correct behavior by observing the policies or behavioral trajectories of one or more experts, has predominantly been accomplished by *inverse reinforcement learning* (IRL) [15], [1]. IRL algorithms generally compute a reward function that "explains" the observed trajectories (typically, by maximally differentiating them from random behavior). Complete discussion of the many types of IRL algorithms is beyond the scope of this paper.

The proposed approach bears some resemblance to IRL, particularly in its inputs (sets of finite behavioral trajectories). Instead of computing a reward function based on the observed trajectories, however, the proposed approach computes a *formula in linear temporal logic* that optimally "explains" the data. This addresses the criticisms of [2], who claim that IRL is insufficient in morally and socially important domains because (1) reward functions can be difficult for human instructors to understand and correct, and (2) some moral and social goals may be too temporally complex to be representable using reward functions.

B. Stochastic Planning with Temporal Logic Specifications

There has been a wealth of work in recent years on providing agents in stochastic domains (namely, Markov Decision Processes) with specifications in linear temporal logic (LTL). The most straightforward approach is [5], which we describe further in section III-C. The problem is to compute some policy which satisfies some LTL formula with maximal probability.

More sophisticated approaches consider the same problem in the face of uncertain transition dynamics [21], [7], partial observability [19], [18], and multi-agent domains [13], [10]. Also relevant to the proposed approach is the idea of "weighted skipping" that appears (in deterministic domains) in [17], [20], [12].

The problem of inferring LTL specifications from behavior trajectories is complementary to the problem of stochastic planning with LTL specifications, much as IRL is complementary to "traditional" reinforcement learning (RL). Specifications learned using the proposed approach may be used for planning, and trajectories generated from planning agents may be used to infer the underlying LTL specification.

C. Inferring Temporal Logic Rules from Agent Behavior

The task of generating temporal logic rules that describe data is not a new one. Automatic identification of temporal logic rules describing the behavior of software programs (in the category of "specification mining") has been attempted in, e.g., [8], [9], [14]. Lemieux et al's Texada [14] allows users to enter custom templates for formulas and retrieves all formulas satisfied by the observed traces up to user-defined support and confidence thresholds; this differs from the work of Gabel and Su, who decompose complex specifications into combinations of predefined templates. Specifications in a temporal logic (rPSTL) have also been inferred from data in continuous control systems in [11]. Each approach deals with (deterministic) program traces.

The proposed approach is most strongly influenced by [4], which casts the task of inferring temporal logic specifications for finite state machines as a multiobjective optimization problem amenable to genetic programming. Much of our approach follows from this work; our novel contribution is introducing the problem of applying such methods to agent behavior in stochastic domains, and in particular our notion of the *violation cost* as an objective function.

III. Preliminaries

In this section we provide formal definitions of Markov Decision Processes (MDPs) and linear temporal logic (LTL); we then outline the approach taken in [5] for planning to satisfy (with maximum probability) LTL formulas in MDPs.

A. Markov Decision Processes

The proposed approach pertains to agents in Markov Decision Processes (MDPs) augmented with a set, Π, of atomic propositions. Since reward functions are not important to this problem, we omit them. All notation and references to MDPs in this paper assume this construction.

Formally, a Markov Decision Process is a tuple

$$\mathcal{M} = \langle S, U, A, P, s_0, \Pi, \mathcal{L} \rangle$$

where

- S is a (finite) set of states;
- U is a (finite) set of actions;
- $A : S \to 2^U$ specifies which actions are available in each state;
- $P : S \times U \times S \to [0, 1]$ is a transition function, with $P(s, a, s') = 0$ if $a \notin A(s)$, so that $P(s, a, s')$ is the probability of transitioning to s' by beginning in s and taking action a;
- s_0 is an initial state;
- Π is a set of atomic propositions; and
- $\mathcal{L} : S \to 2^\Pi$ is the labeling function, so that $\mathcal{L}(s)$ is the set of propositions that are true in state s.

A *trajectory* in an MDP specifies the path of an agent through the state space. A *finite trajectory* is a finite sequence of state-action pairs followed by a final state (e.g., $\tau = (s_0, a_0), \cdots, (s_{T-1}, a_{T-1}), s_T$); an *infinite trajectory* takes $T \to \infty$, and is an infinite sequence of state-action pairs (e.g., $\tau = (s_0, a_0), (s_1, a_1), \cdots$). A sequence (finite or infinite) is only a trajectory if $P(s_t, a_t, s_{t+1}) > 0$ for all $t \in \{0, \cdots, T-1\}$. We will denote by $\text{Traj}_\mathcal{M}$ the set of all finite trajectories in an MDP $\mathcal{M}$, and by $\text{ITraj}_\mathcal{M}$ the set of all infinite trajectories in $\mathcal{M}$. We will denote by $\tau|_T$ the T-time step truncation $(s_0, a_0), \cdots, (s_{T-1}, a_{T-1}), s_T$ of an infinite trajectory $r = (s_0, a_0), (s_1, a_1), \cdots$.

A *policy* $M : \text{Traj}_\mathcal{M} \times U \to [0, 1]$ is a probability distribution over an agent's next action, given its previous (finite) trajectory. A policy is said to be *deterministic* if, for each trajectory, the returned distribution allots nonzero probability for only one action; we write $M : \text{Traj}_\mathcal{M} \to U$. A policy is said to be *stationary* if the returned distribution depends only on the last state of the trajectory; we write $\pi : S \times U \to [0, 1]$.

We denote $\text{ITraj}^M_\mathcal{M}$ the set of all infinite trajectories that may occur under a given policy M. More formally,

$$\text{ITraj}^M_\mathcal{M} = \{\tau = (s_0, a_0), (s_1, a_1), \cdots \in \text{ITraj}_\mathcal{M} : M(\tau|_T, a_T) > 0 \text{ for all } T\}$$

B. Linear Temporal Logic

Linear temporal logic (LTL) [16] is a multimodal logic over propositions that linearly encodes time. Its syntax is as follows:

$$\phi ::= \top \mid \bot \mid p, \text{where } p \in \Pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \to \phi_2 \mid \mathsf{X}\phi \mid \mathsf{G}\phi \mid \mathsf{F}\phi \mid \phi_1 \,\mathsf{U}\, \phi_2$$

Here $\mathsf{X}\phi$ means "in the next time step, ϕ"; $\mathsf{G}\phi$ means "in all present and future time steps, ϕ"; $\mathsf{F}\phi$ means "in some present or future time step, ϕ"; and $\phi_1 \,\mathsf{U}\, \phi_2$ means "ϕ_1 will be true until ϕ_2 holds".

The truth-value of an LTL formula is evaluated over an infinite sequence of *valuations* $\sigma_0, \sigma_1, \cdots$, where for all i, $\sigma_i \subseteq \Pi$. We say $\sigma_0, \sigma_1, \cdots \vDash \phi$ if ϕ is true given the infinite sequence of valuations $\sigma_0, \sigma_1, \cdots$.

There is thus a clear mapping between infinite trajectories and LTL formulas. We abuse notation slightly and define

$$\mathcal{L}((s_0, a_0), (s_1, a_1), \cdots) = \mathcal{L}(s_0), \mathcal{L}(s_1), \cdots$$

We abuse notation further and say that for any $\tau \in \text{ITraj}_\mathcal{M}$, $\tau \vDash \phi$ if $\mathcal{L}(\tau) \vDash \phi$.

We define the probability that a given policy satisfies an LTL formula ϕ by

$$\Pr_{\mathcal{M}}^{M}(\phi) = \Pr\{\tau \in \mathrm{ITraj}_{\mathcal{M}}^{M} : \tau \vDash \phi\}$$

That is, the probability that an infinite trajectory under M will satisfy ϕ.

Each LTL formula can be translated into a *deterministic Rabin automaton* (DRA), a finite automaton over infinite words. DRAs are the standard approach to model checking for LTL. A DRA is a tuple

$$\mathcal{D} = \langle Q, \Sigma, \delta, q_0, F \rangle$$

where

- Q is a finite set of states;
- Σ is an alphabet (in this case, $\Sigma = 2^{\Pi}$, so words are infinite sequences of valuations);
- $\delta : Q \times \Sigma \to Q$ is a (deterministic) transition function;
- q_0 is an initial state; and
- $F = \{(\mathrm{Fin}_1, \mathrm{Inf}_1), \cdots, (\mathrm{Fin}_k, \mathrm{Inf}_k)\}$, where $\mathrm{Fin} \subseteq Q$, $\mathrm{Inf} \subseteq Q$ for all $(\mathrm{Inf}, \mathrm{Fin}) \in F$ specifies the acceptance conditions.

A *run* $r = q_0, q_1, \cdots$ of a DRA is an infinite sequence of DRA states such that there is some word $\sigma_0\sigma_1\cdots$ such that $\delta(q_i, \sigma_i) = q_{i+1}$ for all i. A run r is considered accepting if there exists some $(\mathrm{Fin}, \mathrm{Inf}) \in F$ such that for all $q \in \mathrm{Fin}$, q is visited only finitely often in r, and Inf is visited infinitely often in r.

C. Stochastic Planning with LTL Specifications

Planning to satisfy a given LTL formula ϕ within an MDP $\mathcal{M}$ with maximum probability generally follows the approach of [5].

The planning agent runs the DRA for ϕ alongside $\mathcal{M}$ by constructing a *product MDP* $\mathcal{M}^{\times}$ which augments the state space to include information about the current DRA state.

Formally, the product of an MDP $\mathcal{M} = \langle S, U, A, T, s_0, \Pi, \mathcal{L} \rangle$ and a DRA $\mathcal{D} = \langle Q, 2^{\Pi}, \delta, q_0, F \rangle$ is an MDP

$$\mathcal{M}^{\times} = \langle S^{\times}, U^{\times}, A^{\times}, P^{\times}, s_0^{\times}, \Pi^{\times}, \mathcal{L}^{\times} \rangle$$

where

- $S^{\times} = S \times Q$;
- $U^{\times} = U$; $A^{\times} = A$;
- $P^{\times}((s,q), a, (s',q')) =$

$$\begin{cases} P(s,a,s') & \text{if } q' = \delta(q, \mathcal{L}(s')) \\ 0 & \text{otherwise} \end{cases}$$

- $s_0^{\times} = (s_0, \delta(q_0, \mathcal{L}(s_0)))$
- $\Pi^{\times} = \Pi$; and $\mathcal{L}^{\times} = \mathcal{L}$.

The agent constructs the product MDP $\mathcal{M}^{\times}$, and then computes its *accepting maximal end components* (AMECs). An *end component* $\mathcal{E}$ of an MDP $\mathcal{M}^{\times}$ is a set of states $S_{\mathcal{E}} \subset S^{\times}$ and an action restriction (mapping from states to sets of actions) $A_{\mathcal{E}} : S_{\mathcal{E}} \to 2^{U}$ such that (1) any agent in $S_{\mathcal{E}}$ that performs only actions as specified by $A_{\mathcal{E}}$ will remain in $S_{\mathcal{E}}$; and (2) any agent with a policy assigning nonzero probability to all actions in $A_{\mathcal{E}}$ is guaranteed to eventually visit each state in $A_{\mathcal{E}}$ infinitely often.

An end component thus specifies a set of states $S_{\mathcal{E}}$ such that with an appropriate choice in policy, the agent can guarantee that it will remain in $S_{\mathcal{E}}$ forever, and that it will reach every state in $S_{\mathcal{E}}$ infinitely often. An end component is maximal if it is not a proper subset of another end component. An end component is *accepting* if there is some $(\mathrm{Fin}, \mathrm{Inf}) \in F$ such that (1) if $q \in \mathrm{Fin}$, then $(s,q) \notin S_{\mathcal{E}}$ for all $s \in S$; and (2) there exists some $q \in \mathrm{Inf}, s \in S$ such that $(s,q) \in S_{\mathcal{E}}$. In this case, by entering $S_{\mathcal{E}}$ and choosing an appropriate policy (for instance, a uniformly random policy over $A_{\mathcal{E}}$), the agent guarantees that the DRA run will be accepting. A method for computing the AMECs of the product MDP is found in [3].

The problem of satisfying ϕ with maximal probability is thus reduced to the problem of reaching, with maximal probability, any state in any AMEC. [5] shows how this can be solved using linear programming.

IV. Optimization Problem

Suppose that an agent is given some set of finite behavior trajectories $\tau^1, \cdots, \tau^m \in \mathrm{Traj}_{\mathcal{M}}$, where $\tau^i = (s_0^i, a_0^i), \cdots, (s_{T_i-1}^i, a_{T_i-1}^i), s_{T_i}^i$ for all $i \in \{1, \cdots, m\}$.

We refer to the agent whose trajectories are observed as the *demonstrator*, and the agent that observes the trajectories as the *apprentice*. There may be several demonstrators satisfying the same objectives; this does not affect the proposed approach.

The proposed problem is to infer an LTL specification that well (and succinctly) explains the observed trajectories. This can be cast as a multiobjective optimization problem with two objective functions:

1) An objective function representing how well a candidate LTL formula explains the observed trajectories (and distinguishes them from random behavior); and
2) An objective function representing the complexity of a candidate LTL formula.

This section proceeds by describing a notion of "violation cost" (and defining the violation cost of infinite trajectories and policies) and using it to define two alternate objective functions representing (a) how well a candidate formula explains the *actual observed state sequence* (a "state-based" objective function), and (b) how well a candidate formula explains the *actions* of the demonstrator in each state (an "action-based" objective function). We then describe the simple notion of formula complexity we will utilize, and formulate the optimization problem.

A. Violation Cost

We are interested in computing LTL formulas that well explain the demonstrator's trajectories. These formulas should be satisfied by the observed behavior, but not by random behavior within the same MDP (since, for example, the trivial formula $\mathsf{G}\ \top$ will be satisfied by the observed behavior, but also by random behavior). Ideally we could assign a

"cost" either to trajectories (finite or infinite) or to policies (and, particularly, to the uniformly random policy in $\mathcal{M}$), where the cost of a trajectory or policy corresponds to its adherence to or deviance from the specification. Given such a cost function C, the objective would be to minimize $\sum_i \big(C(\tau^i) - C(\pi_{rand})\big)$, where $\pi_{rand} : S \times U \to [0,1]$ is the uniformly-random (stationary) policy over $\mathcal{M}$:

$$\pi_{rand}(s,a) = \begin{cases} \frac{1}{|A(s)|} & \text{if } a \in A(s) \\ 0 & \text{otherwise} \end{cases}$$

The obvious choice of such a cost function (over infinite trajectories τ) would be the indicator function $\mathbf{1}_{\tau \not\models \phi}$ which returns 0 if $\tau \models \phi$ and 1 otherwise; this function may be extended to general policies M by $1 - \Pr^M_{\mathcal{M}}(\phi)$. This function, however, cannot distinguish between small and large deviances from the specification. For example, given the specification $\mathsf{G}\ p$, this function cannot differentiate between τ such that p is *almost always* true and τ such that p is *never* true. We thus propose a more sophisticated cost function.

For $\tau \in \mathrm{ITraj}_{\mathcal{M}}$, N a set of nonnegative integers, we define $\tau \backslash N$ to be the subsequence of τ omitting the state-action pairs with time step indices in N. For example, $(s_0,a_0),(s_1,a_1),(s_2,a_2),,\cdots\backslash\{1\} = (s_0,a_0),(s_2,a_2),\cdots$. Each time step with an index in N is said to be "skipped".

We define the *violation cost* of an infinite trajectory $\tau \in \mathrm{ITraj}_{\mathcal{M}}$ subject to the formula ϕ as the (discounted) minimum number of time steps that must be skipped in order for the agent to satisfy the formula:

$$\mathrm{Viol}_\phi(\tau) = \min_{\substack{N \subseteq \mathbb{N}_0 \\ \tau \backslash N \models \phi}} \sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{t \in N} \tag{1}$$

Note that if $\tau \models \phi$, then $\mathrm{Viol}_\phi(\tau) = 0$.

In order to define a similar measure for policies, we must construct an augmented product MDP $\mathcal{M}^\otimes$, which is similar to $\mathcal{M}^\times$ as described in section III-C, but allows an agent to "skip" states by performing at each time step (simultaneously with their normal actions), a "DRA action" $\tilde{a} \in \{keep, susp\}$, where $keep$ causes the DRA to transition as usual, and $susp$ causes the DRA to not update in response to the new state.

Formally, given an MDP $\mathcal{M} = \langle S, U, A, T, s_0, \Pi, \mathcal{L}\rangle$ and a DRA $\mathcal{D} = \langle Q, \Sigma, \delta, q_0, F\rangle$ corresponding to the specification ϕ, we may construct a product MDP $\mathcal{M}^\otimes = \langle S^\otimes, U^\otimes, A^\otimes, T^\otimes, s^\otimes_{-1}, \Pi^\otimes, \mathcal{L}^\otimes\rangle$ as follows:

- $S^\otimes = (S \cup \{s_{-1}\}) \times Q$
- $U^\otimes = (U \cup \{a_{-1}\}) \times \tilde{U}$, where $\tilde{U} = \{keep, susp\}$
- $A^\otimes((s,q)) = \begin{cases} \{a_{-1}\} \times \tilde{U} & \text{if } s = s_{-1} \\ A(s) \times \tilde{U} & \text{otherwise} \end{cases}$
- $s^\otimes_{-1} = (s_{-1}, q_0)$
- $P^\otimes(s^\otimes_{-1}, (a_{-1}, keep), (s_0, \delta(q_0, \mathcal{L}(s_0)))) = 1$
- $P^\otimes(s^\otimes_{-1}, (a_{-1}, susp), (s_0, q_0)) = 1$
- Otherwise, $P^\otimes((s,q),(a,\tilde{a}),(s',q')) =$

$$\begin{cases} P(s,a,s') & \text{if } q' = \delta(q, \mathcal{L}(s')) \text{ and } \tilde{a} = keep \\ P(s,a,s') & \text{if } q' = q \text{ and } \tilde{a} = susp \\ 0 & \text{otherwise} \end{cases}$$

- $\Pi^\otimes = \Pi, \mathcal{L}^\otimes = \mathcal{L}$

The state s_{-1} and action a_{-1} are added so that the agent may choose to "skip" time step $t = 0$. This is necessary for the case that s_0 violates the specification.

Note that the transition dynamics of $\mathcal{M}^\otimes$ are such that N (the set of "skipped" time step indices) can be defined as

$$N = \{t \in \mathbb{N}_0 : \tilde{a}_{t-1} = susp\} \tag{2}$$

Define the *transition cost* $s^\otimes, (a,\tilde{a}), s^{\otimes\prime}$ in $\mathcal{M}^\otimes$ as

$$TC(s^\otimes, (a,\tilde{a}), s^{\otimes\prime}) = \mathbf{1}_{\tilde{a} = susp} \tag{3}$$

The violation cost of a (non-product) trajectory τ can then be rewritten as a discounted sum of the transition costs at each stage, minimized over the DRA actions $\tilde{a}_{-1}, \tilde{a}_0, \tilde{a}_1, \cdots$, subject to the constraint that the DRA run from carrying out τ and the DRA actions must be accepting. This indicates that the violation cost of a policy π may be thought of as the state-value function for the policy π with respect to TC. Indeed, we will define the violation cost of a policy this way.

We define a *product policy* to be a stationary policy $\pi^\otimes : S^\otimes \times (U \cup \{a_{-1}\}) \to [0,1]$. When we consider the violation cost of a policy, we will assume a product policy of this form.

There are two reasons for this. First, when evaluating a candidate specification, we wish to assume the demonstrator had knowledge of that specification (or else we would be unable to notice complex temporal patterns in agent behavior), and thus that the demonstrator's policy is over product states. Second, we wish to allow the demonstrator to observe the new (non-product) state s_t before deciding whether to "skip" time step t. That is, s_t should be observed before $\tilde{a}_{t-1}$ is chosen, which is inconsistent with the typical policy $\pi : S^\otimes \times U^\otimes \to [0,1]$ over the product space.

We can easily construct a product policy from the uniformly random policy on $\mathcal{M}$. We define $\pi^\otimes_{rand}((s,q),a) = \pi_{rand}(s,a)$ for all $s \in S, a \in A$.

Upon constructing the product MDP $\mathcal{M}^\otimes$, we compute its AMECs (as in section III-C). Then let $S_{good} = \bigcup_{i \in \{1,\cdots,p\}} S_{\mathcal{E}_i}$, and let S_{bad} be the set of states in the product space from which no state in S_{good} can be reached; these can be determined by breadth-first search.

We can use a form of the Bellman update equation to perform policy evaluation on a product policy $\pi^\otimes$. For each state $s^\otimes \in S_{bad}$, we initialize the cost of this state to the maximum discounted cost, $\frac{1}{1-\gamma}$, and we do not update these costs. This is done to enforce the constraint that the minimization should be over accepting DRA runs. Otherwise, the violation cost will always be trivially zero (since $\tilde{a} = keep$ will always be picked). The update equation has the following form:

$$\mathrm{Viol}^{(k+1)}((s,q)) \leftarrow \Big(\sum_{a \in A(s)} \pi^\otimes((s,q),a) \Big(\sum_{s' \in S} P(s,a,s') \min\{1 + \gamma \mathrm{Viol}^{(k)}(s',q), \gamma \mathrm{Viol}^{(k)}(s', \delta(q, \mathcal{L}(s')))\} \Big) \Big) \tag{4}$$

Algorithm 1 Best DRA state sequence for finite state sequence $s_0, \cdots, s_T$

1: **function** GETRABINSTATESEQUENCE($Viol_\phi^{\pi^\otimes_{rand}}$, $\mathcal{M}^\otimes$, S_{bad}, $s_0, \cdots, s_T$)
2: $C_t[s^\otimes] \leftarrow \infty$ for all $t \in \{-1, 0, \cdots, T\}, s^\otimes \in S^\otimes$
3: $R_{-1} = \{s^\otimes_{-1}\}$
4: $C_{-1}[s^\otimes_{-1}] \leftarrow 0$
5: $seq_{-1}[s^\otimes_{-1}] \leftarrow q_0$
6: **for** $t \in \{0, \cdots, T\}$ **do**
7: $R_t = \emptyset$
8: **for** $(s, q) \in R_{t-1}$ **do**
9: $q' \leftarrow \delta(q, \mathcal{L}(s_t))$
10: $R_t \leftarrow R_t \cup \{(s_t, q), (s_t, q')\}$
11: **if** $C_{t-1}[(s, q)] + \gamma^t < C_t[(s_t, q)]$ **then**
12: $C_t[(s_t, q)] \leftarrow C_{t-1}[(s, q)] + \gamma^t$
13: $seq_t[(s_t, q)] \leftarrow (seq_{t-1}[(s, q)], q)$
14: **end if**
15: **if** $C_{t-1}[(s, q)] < C_{t-1}[(s_t, q')]$ **then**
16: $C_t[(s_t, q')] \leftarrow C_{t-1}[(s, q)]$
17: $seq_t[(s_t, q')] \leftarrow (seq_{t-1}[(s, q)], q')$
18: **end if**
19: **end for**
20: **end for**
21: $s^\otimes_T \leftarrow \underset{s^\otimes \in R_T \setminus S_{bad}}{\operatorname{argmin}} C_T[s^\otimes] + \gamma^{T+1}\mathrm{Viol}_\phi^{\pi^\otimes_{rand}}(s^\otimes)$ **return** $seq_T[s^\otimes_T]$, $C_T[s^\otimes_T] + \gamma^{T+1}\mathrm{Viol}_\phi^{\pi^\otimes_{rand}}(s^\otimes_T)$
22: **end function**

The $\min\{\cdot\}$ in (4) is where the optimization over $\tilde{a}$ (implicitly) occurs. Choosing $\tilde{a} = susp$ incurs a transition cost of 1 and then causes the DRA to remain in state q; choosing $\tilde{a} = keep$ incurs no transition cost, but causes the DRA to transition to state $\delta(q, \mathcal{L}(s'))$. The ability for the demonstrator to optimize over $\tilde{a}$ after observing the new state s' corresponds to the location of the $\min\{\cdot\}$ in the Bellman update.

We define the violation cost of a policy as the function that results when running this update equation to convergence:

$$\mathrm{Viol}_\phi^{\pi^\otimes}((s, q)) = \lim_{k \to \infty} \mathrm{Viol}^{(k)}((s, q)) \tag{5}$$

We now consider state-based ("what actually happened") and action-based ("what the agent expected to happen") objective functions, for explaining sets of finite trajectories.

The crux of both the state- and action-based objective functions is Algorithm 1. Given a finite sequence of states $s_0, \cdots, s_T$, Algorithm 1 determines the "optimal product-space interpretation" of $s_0, \cdots, s_T$. We define a *product space interpretation* of a sequence of states $s_0, \cdots, s_T$ in an MDP $\mathcal{M}$ as a sequence of DRA states $q_0, \cdots, q_{T+1}$ such that, for all $i \in \{1, \cdots, T+1\}$, either $q_i = q_{i-1}$, or $q_i = \delta(q_{i-1}, \mathcal{L}(s_{i-1}))$. That is, a product-space interpretation specifies a possible trajectory in $\mathcal{M}^\otimes$ that is consistent with the observed trajectory in $\mathcal{M}$.

Algorithm 1 uses dynamic programming to determine, for each time step t, the set of states R_t of DRA states that the demonstrator could be in at time t (lines 3,7, and 10), as well as the minimal violation cost $C_t[q_{t+1}]$ that would need to be accrued in order to be in each such state q_{t+1} (lines 2,4, 12, and 16). The sequence $seq_t[(s_t, q_{t+1})] = q_0, \cdots, q_{T+1}$ that achieves this minimal cost is also computed (lines 5, 13, and 17).

The apprentice then assumes that the demonstrator acted randomly from time step $T+1$ onward. Although this assumption is probably incorrect, it is not entirely unreasonable, since it avoids the assumption that the demonstrator attempted to satisfy the formula after time step $T+1$, which would artificially drive the net violation cost down; this allows the apprentice to reuse values that are already computed in order to evaluate the random policy.

Employing this assumption, the apprentice determines the optimal product-space interpretation as $seq_T[s^\otimes_T]$, where

$$s^\otimes_T = \underset{s^\otimes \in R_T \setminus S_{bad}}{\operatorname{argmin}} C_T[s^\otimes] + \gamma^{T+1}\mathrm{Viol}_\phi^{\pi^\otimes_{rand}}(s^\otimes) \tag{6}$$

1) State-based objective function: We first consider an approach to estimating the violation cost of a finite trajectory that considers only the states visited in the trajectory, ignoring the demonstrator's actions.

The state-based violation cost is the minimand of (6), which is the second value returned by Algorithm 1:

$$\mathrm{Viol}_\phi^S(\tau) = C_T[s^\otimes_T] + \gamma^{T+1}\mathrm{Viol}_\phi^{\pi^\otimes_{rand}}(s^\otimes_T) \tag{7}$$

Thus the state-based objective function for $\tau^1, \cdots, \tau^m$ is the sum of the estimated violation costs of all observed finite trajectories, less m times the expected violation cost of the random policy from the initial state:

$$\mathrm{Obj}^S(\phi) = \left(\sum_{i=1}^m \mathrm{Viol}_\phi^S(\tau^i)\right) - m\mathrm{Viol}_\phi^{\pi^\otimes_{rand}}(s^\otimes_{-1}) \tag{8}$$

The main drawback of the state-based approach is that by ignoring the observed actions, the apprentice neglects a crucial detail: that what the demonstrator "*expected*" or "*intended*" to satisfy may differ from what actually *was* satisfied. The fact that p did not occur does not mean that the demonstrator was not attempting to make p occur with maximal probability, particularly if p is a very rare event. To solve this problem, we consider an action-based approach.

2) Action-based objective function: We now consider estimating the violation cost of a finite trajectory τ by using the observed state-action pairs to compute a partial policy over the product MDP $\mathcal{M}^\otimes$.

To compute the action-based violation cost of a set of trajectories $\tau^1, \cdots, \tau^m$ (Algorithm 2), the apprentice first runs Algorithm 1 to determine the optimal product-space interpretation $q^i_0, \cdots, q^i_{T+1}$ for each trajectory τ^i (line 4), and uses this to compute the resulting product-space sequence $s^{\otimes i}_{-1}, \cdots, s^{\otimes i}_T$ where $s^{\otimes i}_t = (s^i_t, q^i_{t+1})$.

The assumption that for each $i \in \{1, \cdots, m\}$, $t \in \{0, \cdots, T_i - 1\}$, the demonstrator performed a^i_t when in the inferred product MDP state $s^{\otimes i}_t$, induces an action restriction (lines 6 and 11) $A^* : S^\otimes \to 2^U$ where

$$A^*(s^\otimes) = \begin{cases} \bigcup\limits_{\substack{i,t: \\ s^\otimes = s^{\otimes i}_t}} \{a^i_t\} & \text{if } \bigcup\limits_{\substack{i,t: \\ s^\otimes = s^{\otimes i}_t}} \{a^i_t\} \neq \emptyset \\ A^\otimes(s^\otimes) & \text{otherwise} \end{cases}$$

Algorithm 2 Action-based violation cost of set of finite set of finite state-action trajectories $\tau^1, \cdots, \tau^m$ where $\tau^i = (s_0^i, a_0^i), \cdots, (s_{T_i-1}^i, a_{T_i-1}^i), s_{T_i}^i$

1: **function** ACTIONBASEDVIOLATIONCOST($Viol_\phi^{\pi_{\mathcal{M}}^{rand}}$, $\mathcal{M}^\otimes$, S_{bad}, $\tau^1, \cdots, \tau^m$)
2: $A^*[s^\otimes] \leftarrow \emptyset$ for $s^\otimes \in S^\otimes$
3: **for** $i \in \{1, \cdots, m\}$ **do**
4: $q_0^i, \cdots, q_{T_i+1}^i, V \leftarrow$ GETRABINSTATESEQUENCE($s_0^i, \cdots, s_{T_i}^i$)
5: **for** $t \in \{-1, 0, \cdots, T_i - 1\}$ **do**
6: $A^*[(s_t^i, q_{t+1}^i)] \leftarrow A^*[(s_t^i, q_{t+1}^i)] \cup \{a_t^i\}$
7: **end for**
8: **end for**
9: **for** $s^\otimes = (s, q) \in S^\otimes$ **do**
10: **if** $A^*[s^\otimes] = \emptyset$ **then**
11: $A^*[s^\otimes] = A(s)$
12: **end if**
13: **end for**
14: Compute $\text{Viol}_\phi^{\pi_{A^*}^\otimes}$ using (4)
15: **return** $\text{Viol}_\phi^{\pi_{A^*}^\otimes}(s_{-1}^\otimes)$
16: **end function**

The apprentice may then compute, using the Bellman update (4), the violation cost of the policy $\pi_{A^*}^{rand}(s^\otimes)$ that uniformly-randomly chooses an action from $A^*(s^\otimes)$ at each state $s^\otimes$ (line 14):

$$\pi_{A^*}^\otimes(s^\otimes, a) = \begin{cases} \frac{1}{|A^*(s^\otimes)|} & \text{if } a \in A^*(s^\otimes) \\ 0 & \text{otherwise} \end{cases}$$

The action-based objective function is then

$$\text{Obj}^A(\phi) = \text{Viol}_\phi^{\pi_{A^*}^\otimes}(s_{-1}^\otimes) - \text{Viol}_\phi^{\pi_{rand}^\otimes}(s_{-1}^\otimes) \qquad (9)$$

B. Formula Complexity

Given two formulas that equally distinguish between the observed behavior and random behavior, we wish to select the less complex of the two. Here it suffices to simply minimize the number of nodes in the parse tree for the LTL formula (that is, the total number of symbols in the formula). There are also more sophisticated ways to evaluate formula complexity (such as that used in [4]), but they are not necessary for our purposes.

C. Multiobjective Optimization Problem

Given some set of finite trajectories $\tau^1, \cdots, \tau^m$, we thus frame the problem of inferring some LTL formula ϕ that describes $\tau^1, \cdots, \tau^m$ as

$$\min_{\phi \in \text{LTL}} (\text{Obj}(\phi), FC(\phi))$$

where Obj is either Obj^S, as described in (8), or Obj^A, as described in (9); FC is formula complexity (in this case, the number of nodes in the formula) as specified in section IV-B.

V. EXAMPLES

To demonstrate the effectiveness of the proposed objective functions, we employed genetic programming to evolve a set of LTL formulas (where formulas are represented by their parse trees) in two domains. A summary of the domains used is in Table I. In all demonstrations, we used MOEAFramework [**?**] for genetic programming, using standard tree crossover and mutation operations [**?**]. We consider (separately) the state-based and action-based objectives. NSGA-II over each set of objectives was run for 50 generations with a population size of 100. This process was repeated 20 times. We employed BURLAP [**?**] for MDP planning, and Rabinizer 3 [**?**] for converting LTL formulas to DRAs. In each case, we restricted search to formulas of the form $\mathsf{G}\ \phi$.

The tables in this section show formulas that are *Pareto efficient* in at least two NSGA-II runs - that is, there were no solutions within those runs that outperformed them on both objectives. For any Pareto inefficient formula ϕ, there is some formula ϕ' which both (1) better explains the demonstrated trajectories (as measured by the violation-cost objective function) and (2) is simpler. Thus it is reasonable to restrict consideration to only Pareto efficient solutions.

A. SlimChance domain

The SlimChance domain consists of two states: s_{GOOD}, a "good" state, and s_{BAD}, a "bad" state. The agent has two actions: try, and $notry$. If the agent performs $notry$, the next state is always s_{BAD}; if the agent performs try, the next state is s_{GOOD} with small probability $\epsilon = 0.01$ and s_{BAD} otherwise. Thus, performing the try action is "trying" to make the good state occur, but will rarely succeed.

The set Π of atomic propositions for this problem consists of a single proposition *good*, which is true in s_{GOOD} but false in s_{BAD}. We then suppose that the agent is attempting to satisfy the simple LTL formula $\mathsf{G}\ \mathit{good}$.

A demonstrator attempting to minimize violation cost generated three trajectories of 10 time steps each. This resulted in the following trajectories (note that $\tau^1 = \tau^3$, which occurred randomly):

$$\begin{aligned}\tau^1, \tau^3 =& (s_{BAD}, try), (s_{BAD}, try), (s_{BAD}, try), \\ & (s_{BAD}, try), (s_{BAD}, try), (s_{BAD}, try), (s_{BAD}, try), \\ & (s_{BAD}, try), (s_{BAD}, try), (s_{BAD}, try), s_{BAD}\end{aligned}$$

$$\begin{aligned}\tau^2 =& (s_{BAD}, try), (s_{GOOD}, try), (s_{BAD}, try), (s_{BAD}, try), \\ & (s_{BAD}, try), (s_{BAD}, try), (s_{BAD}, try), (s_{BAD}, try), \\ & (s_{BAD}, try), (s_{BAD}, try), (s_{BAD}, try), s_{BAD}\end{aligned}$$

Tables II and III show all solutions that were Pareto efficient in at least two runs, for Obj^S and Obj^A respectively. The results emphasize the distinction between the two objective functions. In Table II the correct formula $\mathsf{G}\ \mathit{good}$ is Pareto efficient in two runs, but in most runs the obviously-incorrect $\mathsf{G}\ \bot$ is the only Pareto efficient formula (and note that $\text{Obj}^S(\mathsf{G}\ \bot) \approxeq \text{Obj}^S(\mathsf{G}\ \mathit{good})$). In contrast, Table III shows that when using Obj^A, the true function $\mathsf{G}\ \mathit{good}$ is Pareto efficient in all twenty runs.

TABLE I

SUMMARY OF EXAMPLE DOMAINS, WITH RUN TIMES FOR STATE-BASED/ACTION-BASED OBJECTIVES

Domain	# States	# Actions	"Actual" specification	Time, state-based (s)	Time, action-based (s)
SlimChance	2	2	$\mathsf{G}\ good$	174.8 ± 18.0	372.0 ± 43.3
CleaningWorld	77	5	$\mathsf{G}\ (\mathsf{X}(vacuum)\ \mathsf{U}\ roomClean)$	19139.3 ± 671.2	32932.3 ± 1755.0

TABLE II

PARETO EFFICIENT SOLUTIONS IN STATE-BASED SLIMCHANCE

Formula ϕ	$\mathrm{Obj}^S(\phi)$	$FC(\phi)$	# Runs
$\mathsf{G}\bot$	-0.3139852	2	18
$\mathsf{G}\ good$	-0.3139852	2	2

TABLE III

PARETO EFFICIENT SOLUTIONS IN ACTION-BASED SLIMCHANCE

Formula ϕ	$\mathrm{Obj}^A(\phi)$	$FC(\phi)$	# Runs
$\mathsf{G}\ good$	-0.4623490	2	20
$\mathsf{G}(good\ \mathsf{U}\ (\mathsf{X}\ good))$	-0.4939355	5	5
$\mathsf{G}(good \vee \mathsf{X}\ good)$	-0.9400473	5	5
$\mathsf{G}((\mathsf{X}\ good)\ \mathsf{U}\ good)$	-0.9400473	5	3
$\mathsf{G}((\mathsf{X}\ good) \vee good)$	-0.9400424	5	2

TABLE IV

PARETO EFFICIENT SOLUTIONS IN STATE-BASED CLEANINGWORLD

Formula ϕ	$\mathrm{Obj}^S(\phi)$	$FC(\phi)$	# Runs
$\mathsf{G}\ roomClean$	-208.69876	2	20
$\mathsf{G}(\mathsf{F}\ roomClean)$	-216.91139	3	20
$\mathsf{G}((\mathsf{X}\ roomClean) \vee vacuum)$	-217.40816	5	2
$\mathsf{G}((\mathsf{G}\ \top)\ \mathsf{U}\ roomClean)$	-216.91169	5	2
$\mathsf{G}(\mathsf{F}(undock\ \mathsf{U}\ roomClean))$	-216.91170	5	2

TABLE V

PARETO EFFICIENT SOLUTIONS IN ACTION-BASED CLEANINGWORLD

Formula ϕ	$\mathrm{Obj}^A(\phi)$	$FC(\phi)$	# Runs
$\mathsf{G}(roomClean)$	-72.74240	2	20
$\mathsf{G}(\mathsf{F}\ roomClean)$	-75.15686	3	20
$\mathsf{G}(vacuum \vee \mathsf{F}\ roomClean)$	-75.15832	5	3
$\mathsf{G}(\mathsf{F}(roomClean \vee dock))$	-75.15782	5	3
$\mathsf{G}((\mathsf{F}\ roomClean) \vee dock)$	-75.15832	5	2
$\mathsf{G}((\mathsf{X}roomClean) \vee vacuum)$	-75.64639	5	2

B. CleaningWorld domain

In the CleaningWorld domain, the agent is a vacuum cleaning robot in a dirty room. The room is characterized by some initial amount $dirt \in \mathbb{N}_0$ of dirt; the agent has some battery level $battery \in \mathbb{N}_0$. The actions available to the agent are: $vacuum$, which reduces both $dirt$ and $battery$ by one; $dock$, which plugs the robot into a charger, allowing it to increment $battery$ for each time step it remains docked; $undock$, which unplugs the robot from the charger; $wait$, which allows the robot to remain docked if it is currently docked, but otherwise simply decrements $battery$. If the robot's battery dies ($battery = 0$), the robot may only perform the dummy action $beDead$. The domain has two propositions $batteryDead$, which is true iff $battery = 0$, and $roomClean$, which is true iff $dirt = 0$. There are also propositions corresponding to each action (where, e.g., the proposition $vacuum$ is true whenever the agent's last action was to vacuum). The agent is to satisfy the LTL objective $\mathsf{G}\ ((\mathsf{X}\ vacuum)\ \mathsf{U}\ roomClean)$.

An agent attempting to minimize violation cost for this specification produced three demonstration trajectories of 10 time steps each. Because CleaningWorld is deterministic, all three trajectories were identical. Here we represent each state s by (d, b) where d is the amount of dirt still in the room in state s and b is the robot's current battery level.

$$\begin{aligned}\tau^1, \tau^2, \tau^3 = &((5,3), vacuum), ((4,2), vacuum),\\ &((3,1), dock), ((3,1), wait), ((3,3), undock),\\ &((3,3), vacuum), ((2,2), vacuum), ((1,1), dock),\\ &((1,1), wait), ((1,3), undock), (1,3)\end{aligned}$$

Tables IV and V show all solutions that were Pareto efficient in at least two runs, for Obj^S and Obj^A respectively. The formulas $\mathsf{G}\ roomClean$ and $\mathsf{G}(\mathsf{F}\ roomClean)$ are generated in all 20 runs by both Obj^S and Obj^A. These formulas (in particular, $\mathsf{G}\ roomClean$) arguably better describe agent behavior than the "actual" specification $\phi_{act} = \mathsf{G}((\mathsf{X}\ vacuum)\ \mathsf{U}\ roomClean)$: they are simpler than ϕ_{act} while generating identical trajectories. This is reflected by the fact that ϕ_{act} *was* generated by the algorithm for both state- and action-based runs, but $\mathrm{Obj}^S(\phi_{act}) = -215.78773$, $\mathrm{Obj}^A(\phi_{act}) = -75.10621$, and $FC(\phi_{act}) = 5$, which is Pareto dominated by $\mathsf{G}(\mathsf{F}\ roomClean)$ when considering either Obj^S or Obj^A. Perhaps because of this, the actual formula is never recovered (although similar formulas occasionally are, such as $\mathsf{G}((\mathsf{X}roomClean) \vee vacuum)$).

VI. DISCUSSION

While for demonstration purposes we chose to use NSGA-II for optimization, in principle any algorithm that can optimize over LTL formulas should suffice. Exploring other algorithms is a topic for future work. In particular, the genetic programming methods employed operate entirely on the syntax of LTL; a method that can make some use of LTL semantics may find optimal solutions more efficiently.

Optimizing over the space of all LTL formulas is difficult because of the combinatorial nature of this space. Since the number of LTL formulas of length ℓ increases exponentially in ℓ, optimization algorithms like NSGA-II are likely to recover simple formulas that explain the demonstrator's behavior reasonably well, but are less likely to recover complex formulas that better explain the behavior.

We do not specify how to select between Pareto efficient solutions; this depends on the relative degree to which system designers value simplicity versus explanatory power. In practice, system designers with clear preferences could convert the given problem into a single-objective problem with objective $f(\mathrm{Obj}(\phi), FC(\phi))$ where f is some nondecreasing function encoding these preferences.

The major drawback of the proposed approach is its scalability. Table I indicates that evaluation on CleaningWorld

with the action-based objective took, on average, roughly 9h 9m. For problems with much larger state and action spaces, this approach is certainly intractable. Theoretically, a single iteration in the computation of Viol_ϕ^π takes time in $O(|S|^2|Q||U|)$. Run time for objective function evaluation also scales linearly in the total number of demonstration time steps. Identifying approaches with better theoretical and practical run times is a topic for future work.

This paper also assumes that the demonstrator is operating in an environment with complete information (e.g., an MDP), no other agents, and known transition dynamics. Extensions to unknown transition dynamics, POMDPs, and multi-agent domains are a topic for future work.

In both given examples, the "true" specification can be modeled using a reward function: in SlimChance, give high reward if and only if the agent is in s_{GOOD}; in CleaningWorld, give high reward only when $roomClean$ is true. IRL may easily recover these reward functions, and would likely converge more quickly than our approach. These examples are meant more to show the viability of the proposed approach than its superiority to IRL in these domains.

While the given problem assumes that the apprentice passively observes the demonstrator's trajectories, future work could consider an active learning approach, in which the apprentice (for example) poses new MDPs involving the same predicates (or perturbs the given MDP), and 'asks' the demonstrator to generate trajectories in the posed MDPs.

VII. Conclusion

In this paper, we introduced the problem of inferring linear temporal logic (LTL) specifications from agent behavior in Markov Decision Processes as a road to interpretable apprenticeship learning, combining the representational power and interpretability of temporal logic with the generalizability of inverse reinforcement learning. We formulated this as a two-objective optimization problem, and introduced objective functions using a notion of "violation cost" to quantify the ability of an LTL formula to explain demonstrated behavior. We presented results using genetic programming to solve this problem in the SlimChance and CleaningWorld domains.

VIII. Acknowledgements

This project was in part supported by ONR grant N00014-16-1-2278.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. *Proc. 21st International Conference on Machine Learning (ICML)*, pages 1–8, 2004.

[2] Thomas Arnold, Daniel Kasenberg, and Matthias Scheutz. Value alignment or misalignment–what will keep systems accountable? In *3rd International Workshop on AI, Ethics, and Society*, 2017.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[4] Daniil Chivilikhin, Ilya Ivanov, and Anatoly Shalyto. Inferring Temporal Properties of Finite-State Machine Models with Genetic Programming. In *Proc. 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1185–1188, 2015.

[5] Xu Chu Ding, Stephen L. Smith, Calin Belta, and Daniela Rus. LTL control in uncertain environments with probabilistic satisfaction guarantees. In *Proceedings - IFAC World Congress*, volume 18, pages 3515–3520, 2011.

[6] Juraj Dzifcak, Matthias Scheutz, Chitta Baral, and Paul Schermerhorn. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4163–4168, 2009.

[7] Jie Fu and Ufuk Topcu. Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints. In *Robotics: Science and Systems X*, 2014.

[8] Mark Gabel and Zhendong Su. Symbolic mining of temporal specifications. In *Proc. 30th International Conference on Software Engineering*, ICSE '08, pages 51–60, New York, NY, USA, 2008. ACM.

[9] Mark Gabel and Zhendong Su. Online inference and enforcement of temporal properties. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 15–24, New York, NY, USA, 2010. ACM.

[10] M. Guo and D. V. Dimarogonas. Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research*, 34(2):218–235, 2014.

[11] Zhaodan Kong, Austin Jones, Ana Medina Ayala, Ebru Aydin Gol, and Calin Belta. Temporal Logic Inference for Classification and Prediction from Data. *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, pages 273–282, 2014.

[12] Morteza Lahijanian, Shaull Almagor, Dror Fried, Lydia E Kavraki, and Moshe Y Vardi. This Time the Robot Settles for a Cost: A Quantitative Approach to Temporal Logic Planning with Partial Satisfaction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, pages 3664–3671, 2015.

[13] Kevin Leahy, Austin Jones, Mac Schwager, and Calin Belta. Distributed Information Gathering Policies under Temporal Logic Constraints. In *IEEE Conference on Decision and Control (CDC)*, volume 54, pages 6803–6808, 2015.

[14] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General ltl specification mining. In *Automated Software Engineering (ASE), 30th IEEE/ACM International Conference on*, pages 81–92. IEEE, 2015.

[15] Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proc. Seventeenth International Conference on Machine Learning*, volume 0, pages 663–670, 2000.

[16] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[17] Luis I. Reyes Castro, Pratik Chaudhari, Jana Tümová, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Incremental sampling-based algorithm for minimum-violation motion planning. In *Proc. IEEE Conference on Decision and Control*, pages 3217–3224, 2013.

[18] Rangoli Sharan and Joel Burdick. Finite state control of POMDPs with LTL specifications. In *Proceedings of the American Control Conference*, pages 501–508, 2014.

[19] Mária Svoreňová, Martin Chmelík, Kevin Leahy, Hasan Ferit Eniser, Krishnendu Chatterjee, Ivana Černá, and Calin Belta. Temporal logic motion planning using POMDPs with parity objectives. In *Proceedings of the 18th International Conference on Hybrid Systems Computation and Control*, pages 233–238, 2015.

[20] Jana Tumova, Gavin C Hall, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Least-violating control strategy synthesis with safety rules. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2013.

[21] Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Robust control of uncertain Markov Decision Processes with temporal logic specifications. In *IEEE Conference on Decision and Control (CDC)*, volume 51, pages 3372–3379, 2012.