Abstract—We propose to synthesize a control policy for a Markov decision process (MDP) such that the resulting traces of the MDP satisfy a linear temporal logic (LTL) property. We construct a product MDP that incorporates a deterministic Rabin automaton generated from the desired LTL property. The reward function of the product MDP is defined from the acceptance condition of the Rabin automaton. This construction allows us to apply techniques from learning theory to the problem of synthesis for LTL specifications even when the transition probabilities are not known a priori. We prove that our method is guaranteed to find a controller that satisfies the LTL property with probability one if such a policy exists, and we suggest empirically with a case study in traffic control that our method produces reasonable control strategies even when the LTL property cannot be satisfied with probability one.

I. INTRODUCTION

Control of Markov Decision Processes (MDPs) is a problem that is well studied for applications such as robotics surgery, unmanned aircraft control and control of autonomous vehicles [1], [2], [3]. In recent years, there has been an increased interest in exploiting the expressiveness of temporal logic specifications in controlling MDPs [4], [5], [6]. Linear Temporal Logic (LTL) provides a natural framework for expressing rich properties such as stability, surveillance, response, safety and liveness. Traditionally, control synthesis for LTL specifications is solved by finding a winning policy for a game between system requirements and environment assumptions [7], [8].

More recently, there has been an effort in exploiting these techniques in designing controllers to satisfy high level specifications for probabilistic systems. Ding et al. [6] address this problem by proposing an approach for finding a policy that maximizes satisfaction of LTL specifications of the form \( \phi = GF\pi \land \psi \) subject to minimization of the expected cost in between visiting states satisfying \( \pi \). In order to maximize the satisfaction probability of \( \phi \), the authors appeal to results from probabilistic model checking [9], [10]. The methods used for maximizing this probability take advantage of computing maximal end components, which are not well suited for partial MDPs with unknown probabilities. We present a different technique that does not require preprocessing of the model. Our algorithm learns the transition probabilities of a partial model online. Our method can therefore be applied in practical contexts where we start from a partial model with unspecified probabilities.

Our approach is based on finding a policy that maximizes the expected utility of an auxiliary MDP constructed from the original MDP and a desired LTL specification. As in the above mentioned existing work, we convert the LTL specification to a deterministic Rabin automaton (DRA) [11], [12], and construct a product MDP such that the states of the product MDP are pairs representing states of the original MDP in addition to states of the DRA that encodes the desired LTL specification. The novelty of our approach is that we then define a state based reward function on this product MDP based on the Rabin acceptance condition of the DRA. We extend our results to allow unknown transition probabilities and learn them online. Furthermore, we select the reward function on the product MDP so it corresponds to the Rabin acceptance condition of the LTL specification. Therefore, any learning algorithm that optimizes the expected utility can be applied to find a policy that satisfies the specification.

We implement our method using a reinforcement learning algorithm that finds the policy optimizing the expected utility of every state in the Rabin-weighted product MDP. Moreover, we prove that if there exists a strategy that satisfies the LTL specification with probability one, our method is guaranteed to find such a strategy. For situations where a policy satisfying the LTL specification with probability one does not exist, our method finds reasonable strategies. We show this performance for two case studies: 1) Control of an agent in a grid world, and 2) Control of a traffic network with intersections.

This paper is organized as follows: In Section II, we review necessary preliminaries. In Section III-A we define the synthesis problem and provide theoretical guarantees in finding a policy satisfying the specification for a special case. Section III-B discusses a learning approach towards finding an optimal controller. We provide two case studies in Section IV. Finally, we conclude in Section V.

II. PRELIMINARIES

We introduce preliminaries on the specification language and the probabilistic model of the system. We use Linear Temporal Logic (LTL) to define desired specifications. A LTL formula is built of atomic propositions \( \omega \in \Pi \) that are over states of the system that evaluate to True or False, propositional formulas \( \phi \) that are composed of atomic propositions and Boolean operators such as \( \land \) (and), \( \neg \) (negation), and temporal operations on \( \phi \). Some of the
common temporal operators are defined as:

\[
\begin{align*}
G \phi & \quad \phi \text{ is true all future moments.} \\
F \phi & \quad \phi \text{ is true some future moments.} \\
X \phi & \quad \phi \text{ is true the next moment.} \\
\phi_1 U \phi_2 & \quad \phi_1 \text{ is true until } \phi_2 \text{ becomes true.}
\end{align*}
\]

Using LTL, we can define interesting liveness and safety properties such as surveillance properties \(GF\phi\), or stability properties \(FG\phi\).

**Definition 1.** A deterministic Rabin automaton is a tuple \(R = (Q, \Sigma, \delta, q_0, F)\) where \(Q\) is the set of states; \(\Sigma\) is the input alphabet; \(\delta : Q \times \Sigma \rightarrow Q\) is the transition function; \(q_0\) is the initial state and \(F\) represents the acceptance condition: \(F = \{(G_1, B_1), \ldots, (G_{n_P}, B_{n_P})\}\) where \(G_i, B_i \subset Q\) for \(i = 1, \ldots, n_P\).

A run of a Rabin automaton is an infinite sequence \(r = q_0 q_1 \ldots\) where \(q_0 \in Q_0\) and for all \(i > 0\), \(q_{i+1} \in \delta(q_i, \sigma)\), for some input \(\sigma \in \Sigma\). For every run \(r\) of the Rabin automaton, \(\text{inf}(r) \in Q\) is the set of states that are visited infinitely often in the sequence \(r = q_0 q_1 \ldots\). A run \(r = q_0 q_1 \ldots\) is accepting if there exists \(i \in \{1, \ldots, n_P\}\) such that:

\[
\text{inf}(r) \cap G_i \neq \emptyset \quad \text{and} \quad \text{inf}(r) \cap B_i = \emptyset \quad (1)
\]

For any LTL formula \(\phi\) over \(\Pi\), a deterministic Rabin automaton (DRA) can be constructed with input alphabet \(\Sigma = 2^\Pi\) that accepts all and only words over \(\Pi\) that satisfy \(\phi\) [12]. We let \(R_\phi\) denote this DRA.

**Definition 2.** A labeled Markov Decision Process (MDP) is a tuple \(M = (S, A, P, s_0, \Pi, L)\) where \(S\) is a finite set of states of the MDP; \(A\) is a finite set of possible actions (controls) and \(A : S \rightarrow 2^A\) is defined as the mapping from states to actions; \(P\) is a transition probability function defined as \(P : S \times A \times S \rightarrow [0, 1]\); \(s_0 \in S\) is the initial state; \(\Pi\) is a set of atomic propositions; and \(L : S \rightarrow 2^\Pi\) is a labeling function that labels a set of states with atomic propositions.

### III. Synthesis Through Reward Maximization

#### A. Problem Formulation

Consider a labeled MDP

\[
M = (S, A, P, s_0, \Pi, L)
\]

and a linear temporal logic specification \(\phi\).

**Definition 3.** A policy for \(M\) is a function \(\pi : S^+ \rightarrow A\) such that \(\pi(s_0 s_1 \ldots s_n) \in A(s_n)\) for all \(s_0 s_1 \ldots s_n \in S^+\) where \(S^+\) denotes the set of all finite sequences of states in \(S\).

Observe that a policy \(\pi\) for an MDP \(M\) induces a Markov chain which we denote by \(M_\pi\). A run of a Markov chain is an infinite sequence of states \(s_0, s_1, \ldots\), where \(s_0\) is the initial state of the Markov chain, and for all \(i, P(s_i, a, s_{i+1})\) is nonzero for some action \(a \in A\).

Our objective is to compute a policy \(\pi^*\) for \(M\) such that the runs of \(M_\pi^*\) satisfy the LTL formula \(\phi\) with probability one as defined below. Our approach composes \(M\) and the DRA \(R_\phi = (Q, \Sigma, \delta, q_0, F)\) whose acceptance condition corresponds to satisfaction of \(\phi\). We then obtain a policy \(\pi^*\) for this composition. Our approach is particularly amenable to learning-based algorithms as we discuss in Section III-B.

In particular, the policy \(\pi^*\) can be constructed even when the transition probabilities \(P\) for \(M\) are not known. Thus, we present an approach that allows the policy \(\pi^*\) to be found online while learning the transition probabilities of \(M\).

We create a Rabin weighted product MDP \(P\) defined below, using the DRA \(R_\phi\) and labeled MDP \(M\). The set of states \(S_P\) in \(P\) are a set of augmented states with components that correspond to states in \(M\) and components that correspond to states in \(R_\phi\). The set of actions \(A_P\) is identical to the set of actions in \(M\).

To this end, we define a Rabin weighted product MDP given a MDP \(M\) and a DRA \(R\) as follows:

**Definition 4.** A Rabin weighted product MDP or simply product MDP between a labeled MDP \(M = (S, A, P, s_0, \Pi, L)\) and a DRA \(R = (Q, \Sigma, \delta, q_0, F)\) is defined as a tuple \(P = (S_P, A_P, P_P, s_0P, F_P, W_P)\) [6], where:

- \(S_P = S \times Q\) is the set of states.
- \(A_P\) provides the set of control actions from the MDP: \(A_P((s, q)) = A(s)\).
- \(P_P\) is the set of transition probabilities defined as:
  \[
  P_P(s, a, s') = \begin{cases} 
  \frac{P(s, a, s')}{1 - w_G} & \text{if } q' = \delta(q, L(s)) \\
  0 & \text{otherwise}
  \end{cases}
  \]
- \(s_0P = (s_0, q_0)\) is in \(S_P\) and \(s'P = (s', q')\).
- \(s_0 = (s_0, q_0)\) is in \(S_P\) is the initial state.
- \(F_P\) is the acceptance condition given by
  \[
  F_P = \{(G_1, B_1), \ldots, (G_{n_P}, B_{n_P})\}
  \]
  where \(G_i = S \times G_i\) and \(B_i = S \times B_i\).
- For the above acceptance condition, \(W_P = (W_P^i)_{i=1}^{n_P}\) is a collection of reward functions \(W_P^i : S_P \rightarrow \mathbb{R}\) defined by:
  \[
  W_P^i(s) = \begin{cases} 
  w_G & \text{if } s \in G_i \\
  w_B & \text{if } s \in B_i \\
  0 & \text{if } s \in S \setminus (G_i \cup B_i)
  \end{cases}
  \]
  where \(w_G > 0\) is a positive reward, \(w_B < 0\) is a negative reward.

We let \(\Pi_P = \Pi \setminus \Pi_\phi\) denote the product automaton of the corresponding Rabin weighted product MDP [9].

**Definition 5.** A stationary policy \(\pi\) for a product MDP \(P\) is a mapping \(\pi : S_P \rightarrow A_P\) that maps every state to actions selected by policy \(\pi\).

A stationary policy for \(P\) corresponds to a finite memory policy for \(M\). We let \(P_M\) denote the Markov chain induced by applying the stationary policy \(\pi\) to the product MDP \(P\).
Let $r = sp_0sp_1sp_2\ldots$ be a run of $\mathcal{P}_\pi$ with initial product state $sp_0$.

**Definition 6.** Consider a MDP $\mathcal{M}$ and a LTL formula $\phi$ with corresponding DRA $\mathcal{R}_\phi$, let $\mathcal{P}$ be the corresponding Rabin weighted MDP, and let $\pi$ be a stationary policy on $\mathcal{P}$. We say that $\mathcal{M}_\pi$ satisfies $\phi$ with probability one if
\[
\Pr\{s: (s, B_i) \in F_P(s) \}
\]
\[
\inf(s) \cap G_i \neq \emptyset \land \inf(s) \cap B_i = \emptyset\} = 1
\]
where $s$ is a run of $\mathcal{P}_\pi$ initialized at $sp_0$.

Intuitively, $\mathcal{M}_\pi$ satisfies $\phi$ with probability one if the probability measure of the runs of $\mathcal{P}_\pi$ that violate the acceptance condition of $\phi$ is 0.

We let $i$ be index of Rabin acceptance condition for property $\phi$. A reward function $W^i_{\mathcal{P}}(sp)$ on every state is defined in Definition 4 and can be identified by $W^i \in \mathbb{R}^{\vert S_P \vert}$ for some enumeration of $S_P$. We assign a negative reward $w_B$ to states $sp \in B_i = S \times B_i$ since we would like to visit them only finitely often. Similarly, we assign positive rewards $w_0$ to $sp \in G_i$, and reward of 0 on neutral states $sp \in N_i$ to bias the policy towards satisfaction of the Rabin automaton’s acceptance condition.

**Definition 7.** For $i \in \{1, \ldots, n_F\}$, the expected discounted utility for a policy $\pi$ on $\mathcal{P}^i$ with discount factor $0 < \gamma < 1$ is a vector $U^i_\pi = [U^i_\pi(s_0) \ldots U^i_\pi(s_N)]$ for $s_k \in S_P, k \in \{1, \ldots, N\}$ and $N = \vert S_P \vert$, such that:
\[
U^i_\pi = \sum_{n=0}^{\infty} \gamma^n P^n_{\pi} W^i
\]
where $W^i_\pi$ is the vector of the rewards $W^i_{\mathcal{P}}(sp)$ and $P^i_{\pi}$ is a matrix containing the probabilities $P_{\pi}(sp, sp', sp'')$. For simpler notation, we omit the superscript $i$ the index of Rabin acceptance condition of the LTL specification. In the rest of this paper, it is assumed that $W$ and $U_\pi$ are the reward and utility vectors of the product MDP with their corresponding set of Rabin acceptance condition pair $(G_i, B_i)$.

**Definition 8.** A policy that maximizes this expected discounted utility for every state is an optimal policy $\pi^* = [\pi^*(s_0) \ldots \pi^*(s_N)]$, defined as:
\[
\pi^* = \arg \max_{\pi} \sum_{n=0}^{\infty} \gamma^n P^n_{\pi} W
\]

Note that for any policy $\pi$, for all $s \in S_P$ $U_{\pi^*}(s) \leq U_{\pi^*}(s).$ From a product MDP $\mathcal{P}$, we seek a policy that satisfies the LTL specification by optimizing the expected future utility. Note that an optimal policy exists for each acceptance condition $(G_i, B_i) \in F_P$ and thus our reward maximization algorithm must be run on each acceptance condition. The outcome is a collection of strategies $[\pi^*_i]_{i=1}^{n_F}$ where $\pi^*_i$ is the optimal policy under rewards $W^i_{\mathcal{P}}$. We use Definition 6 to determine whether a policy $\pi^*_i$ satisfies $\phi$ with probability one by analyzing properties of the recurrent classes in $\mathcal{P}$ [9].

The following theorem shows that optimizing the expected discounted utility produces a policy $\pi^*$ such that $\mathcal{M}_{\pi^*}$ satisfies $\phi$ with probability one if such a policy exists.

**Theorem 1.** Given MDP $\mathcal{M}$ and LTL formula $\phi$ with corresponding Rabin weighted product MDP $\mathcal{P}$. If there exists a policy $\pi$ such that $\mathcal{M}_{\pi}$ satisfies $\phi$ with probability 1, then there exists $i^* \in \{1, \ldots, n_F\}$, $\gamma^* \in (0, 1]$, and $w_B^* < 0$ such that any algorithm that optimizes the expected future utility of $\mathcal{P}^i$ with $\gamma \geq \gamma^*$ and $w_B \leq w_B^*$ will find such a policy.

Proof. Proof of theorem 1 can be found in Appendix A. Intuitively, choosing $\gamma$ i.e. the discount factor close to 1 enforces visiting $G_i$ infinitely often, and a large enough negative reward $w_B$ enforces visiting $B_i$ only finitely often. This will result in satisfaction of $\phi$ by our algorithm.

Theorem 1 provides a practical approach to synthesizing a control policy $\pi^*$ for the MDP $\mathcal{M}$. After constructing the corresponding product MDP $\mathcal{P}$, a collection of policies $[\pi^*_i]_{i=1}^{n_F}$ is computed that optimize the expected future utility of each $\mathcal{P}^i$. Provided that $\gamma$ and $|w_B|$ are sufficiently large, if there exists a policy $\pi$ such that $\mathcal{M}_{\pi}$ satisfies $\phi$ with probability 1, then for at least one of the computed policies $\pi^*_i, \mathcal{M}_{\pi^*_i}$ satisfies $\phi$ with probability 1. Determining which of the policies satisfy $\phi$ with probability 1 is easily achieved by computing strongly connected components of the resulting Markov chains, for which there exist efficient graph theoretic algorithms [9].

In this section, we have not provided an explicit method for optimizing the expected utility of the product MDP $\mathcal{P}$. If the transition probabilities of $\mathcal{M}$ are not known a priori, then the optimization algorithm must simultaneously learn the transition probabilities while optimizing the expected utility, and tools from learning theory are well-suited for this task. In the following section, we discuss how these tools apply to the policy synthesis problem above.

**B. Synthesis through Reinforcement Learning**

By translating the LTL synthesis problem into an expected reward maximization framework in section 3, it is now possible to use standard techniques in the reinforcement learning literature to find satisfying control policies.

In the previous section, we did not provide an explicit method for optimizing the expected utility of the product MDP $\mathcal{P}$. If the transition probabilities of $\mathcal{M}$ are not known a priori, then the optimization algorithm must 1) Learn the transition probabilities and 2) Optimize the expected utility. Tools from learning theory are well-suited for this task.

Algorithm [1] below is a modified active temporal difference learning algorithm [13] that accomplishes these goals. It is called after each observed transition and updates a set of persistent variables, which include a table of transition frequencies, state utilities, and the optimal policy that can each be initialized by the user with a priori estimates. The magnitude of the update is determined by a learning rate, $\alpha$. Algorithm 1 is customized to take advantage of the structure.
in \( \mathcal{P} \) to converge more quickly to the actual transition probabilities. Observe that product states corresponding to the same labeled MDP state have the same transition probability structure i.e. \( P_\mathcal{P}(s_p, a, s'_p) = P_\mathcal{P}(s_p, a, s'_p) \) if \( s_p = (s, q) \), \( s'_p = (s, \hat{q}) \), \( s_p = (s', q') \), and \( s'_p = (s', \hat{q}') \), where \( q, q', \hat{q}, \hat{q}' \in Q \), and \( s, s' \in S \). Therefore, every iteration in the product MDP can in fact be used to update the transition probability estimates for all product MDP states that share the same labeled MDP state. Thus, the algorithm uses equivalence classes \( ([s_p], a_p) \), where \( [s_p] = s \times Q = \{ s_p = (s, q) | q \in Q \} \) to more quickly converge to the optimal policy.

Traditionally, temporal difference learning occurs over multiple trials where the initial state is reset after each trial [14]. Similarly, in an online application, where we cannot reset the labeled MDP state, we periodically reset the Rabin component of the product state to \( Q_0 \). For instance, if the LTL formula contains any safety specifications, then a safety violation will make it impossible to reach a state with positive reward in \( \mathcal{P} \). To ensure we obtain a correct control action for every state we introduce a function “ResetConditionMet()” in Algorithm 1 that forces a Rabin state reset whenever a safety violation is detected, or heuristically after a set time interval if liveness properties are not being met. In both case studies, we observed that this reset technique results in Algorithm 1 converging to a satisfying policy.

We note that online learning algorithms on general MDPs do not have hard convergence guarantees to the optimal policy because of the exploitation versus exploration dilemma [13]. A learning agent decides whether to explore or exploit via the exploration function \( f_{exp} \). One possible exploration function for probably approximately correct learning observes transitions and builds an internal model of the transition probabilities. The agent defaults to an exploration mode and only explores if it can learn more about the system dynamics [15].

### IV. Case Studies

#### A. Control of an agent in a grid world

For illustrative purposes, we consider an agent in a 5 × 5 grid world that is required to visit regions labeled A and B infinitely often, while avoiding region C. The LTL specification is given as the following formula:

\[
\text{GFA} \land \text{GFB} \land \text{G} \neg \text{C} \quad (7)
\]

The agent is allowed four actions, where each one expresses a preference for a diagonal direction. An “upper right” action will cause the agent to move right with probability 0.4, up with probability 0.4, and remain stationary with probability 0.2. If a wall is located to the agent’s right then it will move up with probability 0.8, if one is located above then it will move to the right with probability 0.8, and if the agent is in the upper right corner, then it is guaranteed to remain in the same location. The dynamics for the other actions are identical after an appropriate rotation.

---

**Algorithm 1 Temporal Difference Learning for \( \mathcal{M}_\mathcal{P} \)**

**Input:** \( s_p \) Current state of \( \mathcal{P} \).

**Output:** \( a_p \) Current action

**Persistent Values:**
- Utilities \( U_\mathcal{P}(s_p) \) for all states of \( \mathcal{P} \) initialized at 0.
- \( N_{sa}(\|s_p\|, a_p) \) a table of frequency of state, action pairs initialized by the user.
- \( N_{s'|sa}(\|s_p\|, a_p, \|s'_p\|) \) a table of frequency of the outcome of the equivalence class \( \|s'_p\| \) for state, action pairs in the equivalence class \( \|s_p\|, a_p \) initialized by the user.

- Optimal Policy \( \pi^* \) for every state. Initialized at 0.
- \( s_p, a_p \) previous state and action, initialized as null

**if** \( s_p \) is new **then**

\[
U_\mathcal{P}(s_p) \leftarrow W_\mathcal{P}(s_p)
\]

**end if**

**if** ResetConditionMet() is True **then**

\[
s'_p = \text{ResetRabinState}(s_p)
\]

**else if** \( s_p \) is not NULL **then**

\[
N_{sa}(\|s_p\|, a_p) \leftarrow N_{sa}(\|s_p\|, a_p) + 1
\]

\[
N_{s'|sa}(\|s_p\|, a_p, \|s'_p\|) \leftarrow N_{s'|sa}(\|s_p\|, a_p, \|s'_p\|) + 1
\]

**for all** \( t \) **do**

\[
P(\|s\|, a, \|t\|) \leftarrow \frac{N_{s'|sa}(\|s\|, a, \|t\|)}{N_{sa}(\|s\|, a)}
\]

**end for**

\[
U_\mathcal{P}(s_p) \leftarrow \alpha U_\mathcal{P}(s_p) + (1 - \alpha)W_\mathcal{P}(s_p)
\]

\[
+ \gamma \max_{a} \sum_{\sigma} P(s_p, a_p, \sigma)U(\sigma)
\]

\[
\pi^*(s_p) \leftarrow \arg \max_{a \in A_\mathcal{P}(s_p)} \sum_{\sigma} P(s_p, a_p, \sigma)U(\sigma)
\]

**end if**

Choose current action \( a_p^* = f_{exp} \)

\[
s_p = s_p
\]

\[
a_p = a_p
\]
The current queue lengths of each link.

The network is the collection of current subintervals containing link $l$. We accommodate up to $C$ into the set $\{l\}

The vehicles that are forwarded divide among downstream links via turn ratios

Let $x_i$ for $i=1,\ldots,4$ denote the number of vehicles enqueued on link $i$. We consider the following control objective:

$$FG(x_1 \leq 30 \land x_2 \leq 30) \land$$

$$GF(x_3 \leq 10) \land GF(x_4 \leq 10) \land$$

$$G((s_{v_2} \land X(\neg s_{v_2})) \implies (XX(\neg s_{v_2}) \land XXX(\neg s_{v_2})))$$.

In words, (10)–(12) is

(Eventually links 1 and 2 have adequate supply) and

(Infinitely often, links 3 and 4 have short queues) and

(When signal $v_2$ actuates link 4, it does so for a minimum of 3 times steps)

where “adequate supply” means the number of vehicles on links 1 and 2 does not exceed 30 vehicles and thus can always accept incoming traffic, and a queue is “short” if the queue length is less than 10. Condition (12) is a minimum green time for actuation of link 4 at signal 2 and may be necessary if, e.g., there is a pedestrian crosswalk across link 2 which requires at least 45 seconds (three time steps) for safe crossing (recall that $s_{v_2}=1$ when link 2 is actuated). The above condition is encoded in a Rabin automaton with one acceptance pair and 37 states. The Rabin-weighted product MDP contains 11,840 states and rewards corresponding to the one acceptance pair.

In Fig. 3 we explore how our approach can be used to synthesize a control policy. Restating (10)–(12), the control objective requires the two solid traces to eventually remain below the threshold at 30 vehicles and for the two dashed traces to infinitely often move below the threshold at 10 vehicles. Additionally, signal 2 should be red for at least three consecutive time steps whenever it switches from green to red.

Fig. 3(a) shows a naïve control policy that synchronously actuates each link for 3 time steps but does not satisfy $\phi$ since $x_2$ remains above 30 vehicles. If estimates of turn ratios and saturation limits are available, e.g., historical data, then we can obtain a MDP that approximates the true traffic dynamics and determine the optimal control policy for the corresponding Rabin-weighted product MDP. When applied to the true traffic model, the controller greatly outperforms the naïve policy but still does not satisfy $\phi$, as shown in Fig.
(b). However, by modifying this policy via reinforcement learning on the true traffic dynamics, we obtain a controller that empirically often satisfies $\phi$ as seen in Fig. 3(c) (Note that we should not expect $\phi$ to be satisfied for all traces of the MDP or all disturbance inputs as such a controller may not exist).

This example suggests how our approach can be utilized in practice: a “reasonable” controller can be obtained by using a Rabin-weighted MDP generated from approximated traffic parameters. This policy can then be modified online to obtain a control policy that better accommodates existing conditions. Additionally, using a suboptimal controller prior to learning is rarely of serious concern for traffic control as the cost is only increased delay and congestion.

V. CONCLUSION

We have proposed a method for synthesizing a control policy for a MDP such that traces of the MDP satisfy a control objective expressed as a LTL formula. We proved that our synthesis method is guaranteed to return a controller that satisfies the LTL formula with probability one if such a controller exists. We provided two case studies: In the first case study, we utilize the proposed method to synthesize a control policy for a virtual agent in a gridded environment, and in the second case study, we synthesize a traffic signal controller for a small traffic network with two signalized intersections.

The most immediate direction for future research is to investigate theoretical guarantees in the case when the LTL specification cannot be satisfied with probability one. For example, it is desirable to prove or disprove the conjecture that for appropriate weightings in the reward function, our proposed method finds the control policy that maximizes the probability of satisfying the LTL specification. In the event that the conjecture is not true, we wish to identify fragments of LTL for which the conjecture holds. Future research will also explore other application areas such as human-in-the-loop semiautonomous driving. 

APPENDIX

A. Proof of Theorem 7

Proof: Suppose $\bar{\pi}$ satisfies $\phi$ with probability 1, then the set of states of $M_{P,\bar{\pi}}$ written as $MC_{\bar{\pi}}$ can be represented as a disjoint union of $T_{\bar{\pi}}$ transient states and $R_{\bar{\pi}}^i$ closed irreducible sets of recurrent classes [16]:

$$MC_{\bar{\pi}} = T_{\bar{\pi}} \sqcup R_{\bar{\pi}}^1 \sqcup \ldots \sqcup R_{\bar{\pi}}^n$$ (13)

Proposition 1. Policy $\bar{\pi}$ satisfies $\phi$ with probability 1 if and only if there exits $(G_i, B_i) \in F_P$ such that $B_i \subset T_{\bar{\pi}}$ and $R_{\bar{\pi}}^j \cap G_i \neq \emptyset$ for all recurrent classes $R_{\bar{\pi}}^j$.

We omit the proof of Proposition 1, however, it readily follows Definition 6.

Let $\Pi^*$ be the finite set of optimal policies that optimize the expected future utility. We constructively show that for large enough values of $\gamma$, the discount factor and $w_B$, the negative reward on non accepting states, all policies $\pi^* \in \Pi^*$ satisfy $\phi$ with probability 1.

Suppose $\pi^* \in \Pi^*$ does not satisfy $\phi$. Then one of the following two cases must be true:

- **Case 1:** There exists a recurrent class $R_{\pi^*}^j$, such that $R_{\pi^*}^j \cap G_i = \emptyset$. This means with policy $\pi^*$ it is possible to visit $G_i$ only finitely often.

- **Case 2:** There exists $b \in B_i$ such that $b$ is recurrent. That is for some recurrent class of the $M_{P,\pi^*}$, $b \in R_{\pi^*}^j$. This translates to the possibility of visiting a state in $B_i$ infinitely often.

We let $\Pi^* = \Pi_1 \cup \Pi_2$, where $\Pi_1(\Pi_2)$ is the set of optimal policies that do not satisfy $\phi$ by violating Case 1 (Case 2).
Notice that this is not a disjoint union.

In addition, we know that the vector of utilities for any policy \( \pi^* \in \Pi \) is \( U_{\pi^*} \in \mathbb{R}^N \), where \( N = |MC_{\pi^*}| \) is the number of states of \( M_{\pi^*} \):

\[
U_{\pi^*} = \sum_{n=0}^{\infty} \gamma^n P_{\pi^*}^n W
\]  

(14)

In this equation \( U_{\pi^*} = [U_{\pi^*}(s_0) \ldots U_{\pi^*}(s_N)]^\top \) and \( W = [W(s_0) \ldots W(s_N)]^\top \) and \( P_{\pi^*}^n \) is the transition probability matrix with entries \( p_{\pi^*}(s_i, s_j) \) which are the probability of transitioning from \( s_i \) to \( s_j \) using policy \( \pi^* \).

We partition the vectors in equation (14) into its transient and recurrent classes:

\[
\begin{align*}
U_{\pi^*}^{\text{tr}} &= \sum_{n=0}^{\infty} \gamma^n \left[ P_{\pi^*}(T, T) [P_{\pi^*}^{tr1} \ldots P_{\pi^*}^{trm}] \right]^n W_{\pi^*}^{\text{tr}} \\
U_{\pi^*}^{\text{rec}} &= \sum_{n=0}^{\infty} \gamma^n \left[ P_{\pi^*}(R, R) \right]^n W_{\pi^*}^{\text{rec}}
\end{align*}
\]  

(15)

Equation (16) shows that any policy \( \pi^* \) over state \( s \) has a non-positive utility \( U_{\pi^*}(s) \leq 0 \).

In equation (20), \( k_1 = \sum_{j=0}^{i-1} N_j, k_2 = \sum_{j=i+1}^{m} N_j, p_{\pi^*}^{\text{rr}} \) is the vector that corresponds to transition probabilities from \( s \in R_{\pi^*}^k \) to any other state in the same recurrent class using policy \( \pi^* \). \( W_j = [W(s_1^j) \ldots W(s_{N_j}^j)] \) is the vector for the reward values of the recurrent class \( R_{\pi^*}^k \). Since none of these states are in \( G_t \), we conclude that for all elements \( w \in W_j, w \leq 0 \).

\[
U_{\pi^*}(s) = U_{\pi^*}^{\text{tr}} + U_{\pi^*}^{\text{rec}}
\]

\[
\leq \sum_{n=0}^{\infty} \gamma^n p_{\pi^*}^{\text{rr}} \leq 0 \quad \Rightarrow \quad U_{\pi^*}(s) \leq 0
\]  

(21)

We first consider the case that \( s \) is in a recurrent class of \( MC_{\pi} \).

- If \( s \) is in some recurrent class \( s \in R_{\pi}^k \), by proposition 1 \( R_{\pi}^k \cap G_t \neq \emptyset \). Therefore, there is at least one \( s_g \in G_t \) such that \( s_g \in R_{\pi}^k \) and \( s \in R_{\pi}^k \). In addition, we know that all states in \( G_t \) are in the transient class. Therefore the vector of rewards in this recurrent class \( W_j \) as defined previously contains non-negative elements. That is for all elements \( w \in W_j, 0 \leq w \) and there exists at least one \( w_g \in W_j, w < w_g \). \n
\[
0 < \sum_{n=0}^{\infty} \gamma^n p_{\pi^*}^{\text{rr}} \leq 0 \quad \Rightarrow \quad U_{\pi^*}(s) < 0
\]

We have shown that for some \( s \), and any policy \( \pi^* \in \Pi \), \( U_{\pi^*}(s) < U_{\pi}(s) \) which contradicts the optimality assumption of \( \pi^* \) for the case where \( s \in R_{\pi}^k \). Thus, we must have that \( s \) is in a transient class of \( MC_{\pi} \).

- If \( s \) is in a transient class \( s \in T_{\pi} \), we first find a lower bound on \( U_{\pi}(s) \), and show this lower bound can be greater than any positive number for large enough choice of \( \gamma \). Note that at minimum all the states in the transient set of \( \pi \) will have utility of \( w_B < 0 \), that is \( W_{\text{trans}} = W_B = [w_B \ldots w_B] \), and there will be only one state \( s_g \in G_t \) that lives in the recurrent class. That is \( w_g \in W \) has a positive reward.

Proposition 2. For transient states \( t_1, t_2 \in T \), there exists \( N < \infty \) such that:

\[
\sum_{n=0}^{N} p^{n}(t_1, t_2) < N,
\]

that is, the infinite sum is bounded [16].

We assume \( q := |T_{\pi}| \) is the number of transient states. In addition, \( P_{\pi}(R, R) \) is a stochastic matrix with row sum of 1 [16].
\[
\sum_{n=0}^{\infty} \gamma^n P^\pi_{\bar{\pi}}(T, T) W^u + \gamma^n P^\pi_{\bar{\pi}}(R, R) W^{rec} < U^u_{\bar{\pi}}
\]  
(24)

\[
N_1 \|q \times q \| W_B + \sum_{n=0}^{\infty} \gamma^n P^\pi_{\bar{\pi}}(R, R) W^{rec} < U^u_{\bar{\pi}}
\]  
(25)

**Proposition 3.** If \( p^\pi(s, s) \) is the probability of returning to a state \( s \) itself in \( n \) time steps, there exists a lower bound on \( \sum_{n=0}^{\infty} \gamma^n p^\pi(s, s) \). First, there exists \( \bar{n} \) such that \( \bar{p}^\pi(s, s) \) is nonzero and bounded. Thus \( s \) visits itself after \( \bar{n} \) time steps with a nonzero probability.

Also we know \( (p^\pi(s, s))^n < p^{n\bar{n}}(s, s) \). Therefore:

\[
\sum_{n=0}^{\infty} \gamma^n p^\pi(s, s) > \sum_{n=0}^{\infty} \gamma^n p^{n\bar{n}}(s, s)
\]  
(26)

\[
> \sum_{n=0}^{\infty} (\gamma^\bar{n})^n (p^{n\bar{n}}(s, s))^n
\]  
(27)

\[
> \frac{1}{1 - \gamma^\bar{n}} \bar{p}
\]  
(28)

Going back to equation (24), we find a stricter lower bound on the utility of every state \( U^u_{\bar{\pi}}(s) \) using proposition [3]

\[
N_1 w_B + \frac{1}{1 - \gamma^\bar{n}} \bar{m} < U^u_{\bar{\pi}}(s) = U^u_{\bar{\pi}}(s)
\]  
(29)

If \( 0 < N_1 w_B + \frac{1}{1 - \gamma^\bar{n}} \bar{m} \),

\[
\Rightarrow U^u_{\bar{\pi}}(s) < U^u_{\bar{\pi}}(s)
\]  
(30)

Here \( \bar{m} = \max(\bar{M}) \) and \( \bar{M} < P_{\bar{\pi}} \bar{P} W^{rec} \), where \( \bar{P} \) is a block matrix whose nonzero elements are \( \bar{p} \) bounds derived from proposition [3].

For a fixed \( w_B \), we can select a large enough \( \gamma \) so that equation (30) holds for all \( \pi^* \in \Pi_1 \). This condition implies equation (31) which contradicts with optimality of any \( \pi^* \in \Pi_1 \). Therefore, \( \pi^* \) cannot be optimal unless it visits \( \xi_{\bar{\pi}} \) infinitely often.

**Case 2:**

Now we consider case 2, where \( \pi^* \in \Pi_2 \). Here for some \( b \in B_s, \ b \in R^2_s \). In addition, this state is in the transient class of \( \bar{\pi}, b \in T_{\bar{\pi}} \). Using the same procedure as the previous case, we find the following upper bound.

\[
U^u_{\bar{\pi}} > \sum_{n=0}^{\infty} \gamma^n P^\pi_{\bar{\pi}}(T, T) W^u
\]  
(32)

\[
> \sum_{n=0}^{\infty} P^\pi_{\bar{\pi}}(T, T) W^u
\]  
(33)

\[
\Rightarrow U^u_{\bar{\pi}}(b) > N_2 \|q \times q \| W_B
\]  
(34)

\[
\Rightarrow U^u_{\bar{\pi}}(b) > N_2 w_B
\]  
(35)

We know that \( b \) is in the recurrent class while using policy \( \pi^* \). So we can use equation (16) to find a bound on the utility. An upper bound assumes that all the other states in the recurrent class have positive reward of \( w_G \).

\[
U^{rec}_{\pi^*} = \sum_{n=0}^{\infty} \gamma^n P^\pi_{\pi^*}(R, R) W^{rec} \Rightarrow
\]  
(36)

\[
U^{rec}_{\pi^*} (b) \leq \sum_{n=0}^{\infty} \gamma^n w_G + \sum_{n=0}^{\infty} \gamma^n p^n_{\pi^*}(b, b) w_B
\]  
(37)

\[
< w_G \frac{1}{1 - \gamma} + w_B \sum_{n=0}^{\infty} \gamma^n p^n_{\pi^*}(b, b)
\]  
(38)

If the following condition in equation (39) holds, we conclude that for a state \( b \), \( U^u_{\pi^*}(b) < U^u_{\bar{\pi}}(b) \) which violates the optimality of \( \pi^* \).

\[
U^u_{\pi^*}(b) < w_G \frac{1}{1 - \gamma} + w_B \sum_{n=0}^{\infty} \gamma^n p^n_{\pi^*}(b, b) < N_2 w_B < U^u_{\bar{\pi}}(b)
\]  
(39)

We only need to enforce:

\[
w_G \frac{1}{1 - \gamma} + w_B \sum_{n=0}^{\infty} \gamma^n p^n_{\pi^*}(b, b) < N_2 w_B
\]  
(40)

Since there are only a finite number of policies in \( \Pi_2 \), from all policies \( \pi^* \in \Pi_2 \), we can find \( \bar{p} \) such that:

\[
\sum_{n=0}^{\infty} \gamma^n p^n_{\pi^*}(b, b) < \sum_{n=0}^{\infty} \gamma^n \bar{p}
\]  
(41)

Therefore equation (40) can be simplified:

\[
w_G \frac{1}{1 - \gamma} + w_B \sum_{n=0}^{\infty} \gamma^n \bar{p} < N_2 w_B
\]  
(42)

\[
w_G \frac{1}{1 - \gamma} + w_B \frac{1}{1 - \gamma} \bar{p} < N_2 w_B
\]  
(43)

\[
(w_G + w_B \bar{p})(\frac{1}{1 - \gamma}) < N_2 w_B
\]  
(44)

\[
(w_G + w_B \bar{p}) - N_2 w_B (1 - \gamma) < 0
\]  
(45)

We assumed without loss of generality \( w_G = 1 \). For a fixed value of \( \gamma \), we choose \( w_B \) small enough so all \( \pi^* \in \Pi_2 \) satisfy equation (45) and violate the optimality condition.

As a result, any optimal policy must satisfy case 2, which is visiting a state in \( B_t \) only finitely often.

For optimal policies \( \pi^* \in \Pi_1 \cap \Pi_2 \), we need to find \( \gamma \) and \( w_B \) such that both conditions for case 1 and case 2 are satisfied. That is:

\[
\begin{cases}
0 < N_1 w_B (1 - \gamma^\bar{n}) + \bar{M} \\
(1 + w_B \bar{p}) - N_2 w_B (1 - \gamma) < 0
\end{cases}
\]  
(46)

We select a pair of \( \gamma \) and \( w_B \) so the system of equations in (46) is satisfied. This solution can be found as follows:

First, for a small real number \( 0 < \epsilon < \bar{M} \), we select \( w_B^* \) so:

\[
1 + w_B^* \bar{p} \leq -\epsilon
\]  
(47)

Then, \( \gamma^* \) is selected so the following holds:

\[
\max \{-N_1 w_B^* (1 - \gamma^*)^\bar{n}, -N_2 w_B^* (1 - \gamma^*)\} < \epsilon
\]  
(48)

The pair of \( (w_B^*, \gamma^*) \) satisfy equation (46), and as a result
none of the policies $\pi^* \in \Pi^*$ are optimal.

REFERENCES

[1] R. Alterovitz, “The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty,” in In Robotics: Science and Systems, 2007.
[2] S. Temizer, M. J. Kochenderfer, L. P. Kaelbling, T. Lozano-Pérez, and J. K. Kuchar, “Collision avoidance for unmanned aircraft using Markov decision processes,” in AIAA Guidance, Navigation, and Control Conference, Toronto, Canada, 2010.
[3] D. Sadigh, K. Driggs-Campbell, A. Puggelli, W. Li, V. Shia, R. Bajcsy, A. Sangiovanni-Vincentelli, S. Sastry, and S. Seshia, “Data-driven probabilistic modeling and verification of human driver behavior,” in Formal Verification and Modeling in Human-Machine Systems, 2014.
[4] E. Wolff, U. Topcu, and R. Murray, “Robust control of uncertain Markov decision processes with temporal logic specifications,” in Decision and Control (CDC), 2012 IEEE 51st Annual Conference on, Dec 2012, pp. 3372–3379.
[5] M. Lahijanian, S. Andersson, and C. Belta, “A probabilistic approach for control of a stochastic system from ltl specifications,” in Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on, Dec 2009, pp. 2236–2241.
[6] X. Ding, S. Smith, C. Belta, and D. Rus, “Optimal control of LTL decision processes with linear temporal logic constraints,” IEEE Transactions on Automatic Control, vol. 59, no. 6, pp. 1636–1649, 2014.
[7] N. Piterman, A. Pnueli, and Y. Sa’ar, “Synthesis of reactive(1) designs,” in VMCAI, 2006, pp. 364–380.
[8] T. Wongpiromsarn, U. Topcu, and R. Murray, “Receding horizon temporal logic planning,” IEEE Transactions on Automatic Control, vol. 57, no. 11, pp. 2817–2830, Nov 2012.
[9] C. Baier and J.-P. Katoen, Principles of model checking. MIT Press, 2008.
[10] M. Vardi, “Probabilistic linear-time model checking: An overview of the automata-theoretic approach,” in Formal Methods for Real-Time and Probabilistic Systems, ser. Lecture Notes in Computer Science, 1999, vol. 1601, pp. 265–276.
[11] J. Klein and C. Baier, “Experiments with deterministic omega-automata for formulas of linear temporal logic,” in Implementation and Application of Automata. Springer, 2004.
[12] S. Safra, “On the complexity of omega -automata,” in Proceedings of the 29th Annual Symposium on Foundations of Computer Science, ser. SFCS ’88, 1988, pp. 319–327.
[13] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Prentice Hall, 2010.
[14] R. Sutton, “Learning to predict by the methods of temporal differences,” Machine Learning, vol. 3, no. 1, pp. 9–44, 1988.
[15] M. Kearns and S. Singh, “Near-optimal reinforcement learning in polynomial time,” Machine Learning, no. 49, pp. 209–232, 2002.
[16] R. Durrett, Essentials of stochastic processes, 2nd ed., ser. Springer texts in statistics. New York : London: Springer, 2012.