A Learning Based Approach to Control Synthesis of Markov Decision Processes for Linear Temporal Logic Specifications

Dorsa Sadigh, Eric S. Kim, Samuel Coogan, S. Shankar Sastry, Sanjit A. Seshia

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 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 different 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 $\phi = \mathbf{GF}\pi \wedge \psi$ subject to minimization of the expected cost in between visiting states satisfying π. In order to maximize the satisfaction probability of ϕ, 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.

This work is supported in part by NDSEG and NSF Graduate Research Fellowships, NSF grant CCF-1116993 and DOD ONR Office of Naval Research N00014-13-1-0341.

The authors are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, {dsadigh, eskim, scoogan, sseshia, sastry}@eecs.berkeley.edu.

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 policy that satisfies the LTL specification with probability one, our method is guaranteed to find such a policy. 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* ϕ that are composed of atomic propositions and Boolean operators such as $\wedge$ (and), $\neg$ (negation), and *temporal operations* on ϕ. Some of the common temporal operators are defined as:

$\mathbf{G}\phi$	ϕ is true all future moments.
$\mathbf{F}\phi$	ϕ is true some future moments.
$\mathbf{X}\phi$	ϕ is true the next moment.
$\phi_1 \mathbf{U} \phi_2$	ϕ_1 is true until ϕ_2 becomes true.

Using LTL, we can define interesting *liveness* and *safety* properties such as surveillance properties $\mathbf{GF}\phi$, or stability properties $\mathbf{FG}\phi$.

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

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, $\inf(r) \in Q$ 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 $i \in \{1, \ldots, n_F\}$ such that:

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

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

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

$$\mathcal{M} = \langle S, \mathcal{A}, P, s_0, \Pi, L \rangle \tag{2}$$

and a linear temporal logic specification ϕ.

Definition 3. *A* policy *for $\mathcal{M}$ is a function $\pi : S^+ \to A$ such that $\pi(s_0 s_1 \ldots s_n) \in \mathcal{A}(s_n)$ 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 π for an MDP $\mathcal{M}$ induces a Markov chain which we denote by $\mathcal{M}_\pi$. 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 π^* for $\mathcal{M}$ such that the runs of $\mathcal{M}_{\pi^*}$ satisfy the LTL formula ϕ with probability one as defined below. Our approach composes $\mathcal{M}$ and the DRA $\mathcal{R}_\phi = \langle Q, \Sigma, \delta, q_0, F \rangle$ whose acceptance condition corresponds to satisfaction of ϕ. We then obtain a policy π^* for this composition. Our approach is particularly amenable to learning-based algorithms as we discuss in Section III-B. In particular, the policy π^* can be constructed even when the transition probabilities P for $\mathcal{M}$ are not known. Thus, we present an approach that allows the policy π^* to be found *online* while learning the transition probabilities of $\mathcal{M}$.

We create a *Rabin weighted product MDP $\mathcal{P}$*, defined below, using the DRA $\mathcal{R}_\phi$ and labeled MDP $\mathcal{M}$. The set of states $S_\mathcal{P}$ in $\mathcal{P}$ are a set of augmented states with components that correspond to states in $\mathcal{M}$ and components that correspond to states in $\mathcal{R}_\phi$. The set of actions $\mathcal{A}_\mathcal{P}$ is identical to the set of actions in $\mathcal{M}$.

To this end, we define a *Rabin weighted product MDP* given a MDP $\mathcal{M}$ and a DRA $\mathcal{R}$ as follows:

Definition 4. *A* Rabin weighted product MDP *or simply* product MDP *between a labeled MDP $\mathcal{M} = \langle S, \mathcal{A}, P, s_0, \Pi, L \rangle$ and a DRA $\mathcal{R} = \langle Q, \Sigma, \delta, q_0, F \rangle$ is defined as a tuple $\mathcal{P} = \langle S_\mathcal{P}, \mathcal{A}_\mathcal{P}, P_\mathcal{P}, s_{\mathcal{P}0}, F_\mathcal{P}, W_\mathcal{P} \rangle$ [6], where:*

- *$S_\mathcal{P} = S \times Q$ is the set of states.*
- *$\mathcal{A}_\mathcal{P}$ provides the set of control actions from the MDP: $\mathcal{A}_\mathcal{P}((s, q)) = \mathcal{A}(s)$.*
- *$P_\mathcal{P}$ is the set of transition probabilities defined as:*

$$P_\mathcal{P}(s_\mathcal{P}, a, s'_\mathcal{P}) = \begin{cases} P(s, a, s') & \text{if } q' = \delta(q, L(s)) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

  $s_\mathcal{P} = (s, q) \in S_\mathcal{P}$ and $s'_\mathcal{P} = (s', q')$.
- *$s_{\mathcal{P}0} = (s_0, q_0) \in S_\mathcal{P}$ is the initial state,*
- *$F_\mathcal{P}$ is the acceptance condition given by*

$$F_\mathcal{P} = \{(\mathcal{G}_1, \mathcal{B}_1), \ldots, (\mathcal{G}_{n_F}, \mathcal{B}_{n_F})\}$$

  where $\mathcal{G}_i = S \times G_i$ and $\mathcal{B}_i = S \times B_i$.
- *For the above acceptance condition, $W_\mathcal{P} = \{W^i_\mathcal{P}\}_{i=1}^{n_F}$ is a collection of reward functions $W^i_\mathcal{P} : S_\mathcal{P} \to \mathbb{R}$ defined by:*

$$W^i_\mathcal{P}(s_\mathcal{P}) = \begin{cases} w_G & \text{if } s_\mathcal{P} \in \mathcal{G}_i \\ w_B & \text{if } s_\mathcal{P} \in \mathcal{B}_i \\ 0 & \text{if } s_\mathcal{P} \in S \backslash (\mathcal{G}_i \cup \mathcal{B}_i) \end{cases} \tag{4}$$

  where $w_G > 0$ is a positive reward, $w_B < 0$ is a negative reward.
  We let $\mathcal{N}_i = S \backslash (\mathcal{G}_i \cup \mathcal{B}_i)$ for every pair of $(\mathcal{G}_i, \mathcal{B}_i)$.

We use the notation $\mathcal{P}^i$ to denote $\mathcal{P}$ with the specific reward function $W^i_\mathcal{P}$. In seeking a policy π for $\mathcal{M}$ such that $\mathcal{M}_\pi$ satisfies ϕ, it suffices to consider *stationary policies* of the corresponding Rabin weighted product MDP [9].

Definition 5. *A* stationary policy *π for a product MDP $\mathcal{P}$ is a mapping $\pi : S_\mathcal{P} \to A_\mathcal{P}$ that maps every state to actions selected by policy π.*

A stationary policy for $\mathcal{P}$ corresponds to a finite memory policy for $\mathcal{M}$. We let $\mathcal{P}_\pi$ denote the Markov chain induced by applying the stationary policy π to the product MDP $\mathcal{P}$.

Let $r = s_{\mathcal{P}0}s_{\mathcal{P}1}s_{\mathcal{P}2}\dots$ be a run of $\mathcal{P}_\pi$ with initial product state $s_{\mathcal{P}0}$.

Definition 6. *Consider a MDP $\mathcal{M}$ and a LTL formula ϕ with corresponding DRA $\mathcal{R}_\phi$, let $\mathcal{P}$ be the corresponding Rabin weighted MDP, and let π be a stationary policy on $\mathcal{P}$. We say that $\mathcal{M}_\pi$* satisfies ϕ with probability 1 *if*

$$\begin{aligned}&Pr(\{r : \exists(\mathcal{G}_i, \mathcal{B}_i) \in F_\mathcal{P}(s)\\ &\mathit{inf}(r) \cap \mathcal{G}_i \neq \emptyset \wedge \mathit{inf}(r) \cap \mathcal{B}_i = \emptyset\}) = 1\end{aligned}$$

where r is a run of $\mathcal{P}_\pi$ initialized at $s_{\mathcal{P}0}$.

Intuitively, $\mathcal{M}_\pi$ satisfies ϕ with probability one if the probability measure of the runs of $\mathcal{P}_\pi$ that violate the acceptance condition of ϕ is 0.
We let i be *index of Rabin* acceptance condition for property ϕ. A reward function $W^i_\mathcal{P}(s_\mathcal{P})$ on every state is specified in Definition 4 and can be identified by $\mathbf{W}^i \in \mathbb{R}^{|S_\mathcal{P}|}$ for some enumeration of $S_\mathcal{P}$. We assign a negative reward w_B to states $s_\mathcal{P} \in \mathcal{B}_i = S \times B_i$ since we would like to visit them only finitely often. Similarly we assign positive rewards w_g to $s_\mathcal{P} \in \mathcal{G}_i$, and reward of 0 on neutral states $s_\mathcal{P} \in \mathcal{N}_i$ to bias the policy towards satisfaction of the Rabin automaton's acceptance condition.

Definition 7. *For $i \in \{1, \dots, n_F\}$, the* expected discounted utility *for a policy π on $\mathcal{P}^i$ with discount factor $0 < \gamma < 1$ is a vector $\mathbf{U}^i_\pi = [U^i_\pi(s_0) \dots U^i_\pi(s_N)]$ for $s_k \in S_\mathcal{P}, k \in \{1, \dots, N\}$ and $N = |S_\mathcal{P}|$, such that:*

$$\boldsymbol{U}^i_\pi = \sum_{n=0}^{\infty} \gamma^n P^n_\pi \mathbf{W}^i \tag{5}$$

where $\mathbf{W}^i$ is the vector of the rewards $W^i_\mathcal{P}(s_\mathcal{P})$ and P_π is a matrix containing the probabilities $P_\mathcal{P}(s_\mathcal{P}, \pi(s_\mathcal{P}), s'_\mathcal{P})$. 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 $\mathbf{W}$ and $\mathbf{U}_\pi$ are the reward and utility vectors of the product MDP with their corresponding set of Rabin acceptance condition pair $(\mathcal{G}_i, \mathcal{B}_i)$.

Definition 8. *A policy that maximizes this expected discounted utility for every state is an* optimal policy $\boldsymbol{\pi^*} = [\pi^*(s_0) \dots \pi^*(s_N)]$, *defined as:*

$$\boldsymbol{\pi^*} = \arg\max_{\boldsymbol{\pi}} \sum_{n=0}^{\infty} \gamma^n P^n_\pi \mathbf{W} \tag{6}$$

Note that for any policy π, for all $s \in S_\mathcal{P} \quad U_\pi(s) \leq U_{\pi^*}(s)$. 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 $(\mathcal{G}_i, \mathcal{B}_i) \in F_\mathcal{P}$ and thus our reward maximization algorithm must be run on each acceptance condition. The outcome is a collection of strategies $\{\pi^*_i\}^{n_F}_{i=1}$ where π^*_i is the optimal policy under rewards $W^i_\mathcal{P}$. We use Definition 6 to determine whether a policy π^*_i satisfies ϕ 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 π such that $\mathcal{M}_\pi$ satisfies ϕ with probability one if such a policy exists.

Theorem 1. *Given MDP $\mathcal{M}$ and LTL formula ϕ with corresponding Rabin weighted product MDP $\mathcal{P}$. If there exists a policy $\bar{\pi}$ such that $\mathcal{M}_{\bar{\pi}}$ satisfies ϕ with probability 1, then there exists $i^* \in \{1, \dots, n_F\}$, $\gamma^* \in [0, 1)$, 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.*

We direct the readers to [13] for a full proof of theorem 1. Intuitively, choosing γ i.e. the discount factor close to 1 enforces visiting $\mathcal{G}_i$ infinitely often, and a large enough negative reward w_B enforces visiting $\mathcal{B}_i$ only finitely often. This will result in satisfaction of ϕ by our algorithm.

The proof, provided in [13], is by contradiction: we show that any unsatisfactory policy must be suboptimal for γ^* sufficiently close to one and w^*_B sufficiently negative. Such values of γ^* and w^*_B can be found constructively.

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

B. Synthesis through Reinforcement Learning

By translating the LTL synthesis problem into an expected reward maximization framework in section III-A, 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 [14] 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, α. 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 $P_\mathcal{P}(s_\mathcal{P}, a, s'_\mathcal{P}) = P_\mathcal{P}(\hat{s}_\mathcal{P}, a, \hat{s}'_\mathcal{P})$ if $s_\mathcal{P} = (s, q)$, $\hat{s}_\mathcal{P} = (s, \hat{q})$, $s'_\mathcal{P} = (s', q')$, and $\hat{s}'_\mathcal{P} = (s', \hat{q}')$, where $q, q', \hat{q}, \hat{q}' \in Q$, and

Algorithm 1 Temporal Difference Learning for $\mathcal{M}_{\mathcal{P}}$

```
Input: $s'_{\mathcal{P}}$ Current state of $\mathcal{P}$.
Output: $a'_{\mathcal{P}}$ Current action
Persistent Values:
· Utilities $U_{\mathcal{P}}(s_{\mathcal{P}})$ for all states of $\mathcal{P}$ initialized at 0.
· $N_{sa}([\![s_{\mathcal{P}}]\!], a_{\mathcal{P}})$ a table of frequency of state, action pairs initialized by the user.
· $N_{s'|sa}([\![s_{\mathcal{P}}]\!], a_{\mathcal{P}}, [\![s'_{\mathcal{P}}]\!])$ a table of frequency of the outcome of the equivalence class $[\![s'_{\mathcal{P}}]\!]$ for state, action pairs in the equivalence class $([\![s_{\mathcal{P}}]\!], a_{\mathcal{P}})$ initialized by the user.
· Optimal Policy $\pi^*$ for every state. Initialized at 0.
· $s_{\mathcal{P}}, a_{\mathcal{P}}$ previous state and action, initialized as null
if $s'_{\mathcal{P}}$ is new then
    $U_{\mathcal{P}}(s'_{\mathcal{P}}) \leftarrow W^i_{\mathcal{P}}(s'_{\mathcal{P}})$
end if
if ResetConditionMet() is True then
    $s'_{\mathcal{P}}$ = ResetRabinState($s'_{\mathcal{P}}$)
else if $s_{\mathcal{P}}$ is not NULL then
    $N_{sa}([\![s_{\mathcal{P}}]\!], a_{\mathcal{P}}) \leftarrow N_{sa}([\![s_{\mathcal{P}}]\!], a_{\mathcal{P}}) + 1$
    $N_{s'|sa}([\![s_{\mathcal{P}}]\!], a_{\mathcal{P}}, [\![s'_{\mathcal{P}}]\!]) \leftarrow N_{s'|sa}([\![s_{\mathcal{P}}]\!], a_{\mathcal{P}}, [\![s'_{\mathcal{P}}]\!]) + 1$
    for all $t$ that $N_{s'|sa}([\![s]\!], a, [\![t]\!]) \neq 0$ do
        $P([\![s]\!], a, [\![t]\!]) \leftarrow$
                $N_{s'|sa}([\![s]\!], a, [\![t]\!]) / N_{sa}([\![s]\!], a)$
    end for
    $U_{\mathcal{P}}(s_{\mathcal{P}}) \leftarrow$
            $\alpha\, U_{\mathcal{P}}(s_{\mathcal{P}}) + (1-\alpha)[W^i_{\mathcal{P}}(s_{\mathcal{P}})$
            $+ \gamma \max_a \sum_\sigma P(s_{\mathcal{P}}, a_{\mathcal{P}}, \sigma) U(\sigma)]$
    $\pi^*(s_{\mathcal{P}}) \leftarrow \arg\max_{a \in A_{\mathcal{P}}(s_{\mathcal{P}})} \sum_\sigma P(s_{\mathcal{P}}, a_{\mathcal{P}}, \sigma) U(\sigma)$
end if
Choose current action $a'_{\mathcal{P}} = f_{exp}$
$s_{\mathcal{P}} = s'_{\mathcal{P}}$
$a_{\mathcal{P}} = a'_{\mathcal{P}}$
```

$s, s' \in S$. 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 $([\![s_{\mathcal{P}}]\!], a_{\mathcal{P}})$, where $[\![s_{\mathcal{P}}]\!] = s \times Q = \{s_{\mathcal{P}} = (s, q) \mid q \in Q\}$ 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 [15]. 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 [14]. 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 [16].

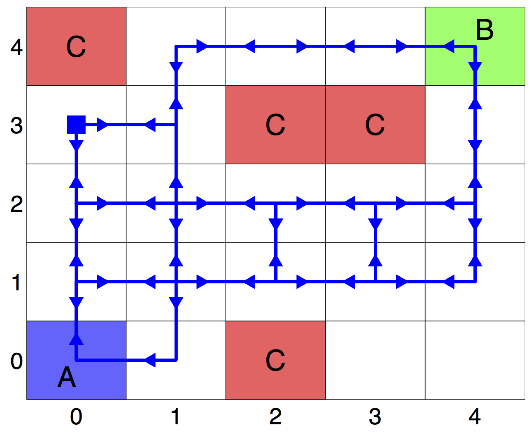

Fig. 1. A grid world example with a superimposed sample trajectory under the policy π^* generated by the reinforcement learning algorithm. The trajectory has a length of 1000 time steps and an initial location (0,3) denoted by a solid square. The arrows denote movement from the box containing the arrow to a corresponding adjacent state. Locations (3,0) and (4,0) do not have any arrows because they are not reachable from the initial state under our policy.

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:

$$\mathbf{GF}A \wedge \mathbf{GF}B \wedge \mathbf{G}\neg C \tag{7}$$

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.

Fig. 1 shows the results of the learning algorithm with an exploration function $f_{exp}(\cdot)$ that simply outputs random actions while learning. The product MDP contained 150 states and one acceptance pair, $\mathcal{G}_i = 500, \mathcal{B}_i = -500$ and $\gamma = 0.98$. There were 600 trials, which are separated by a Rabin reset every 200 time steps.

Observe that no policy exists such that ϕ is satisfied for all runs of the MDP. For example, it is possible that every

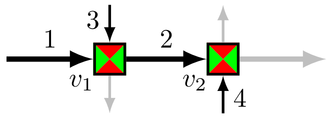

Fig. 2. A traffic network consisting of East-West links 1 and 2 and North-South links 3 and 4 and two signalized intersections. The gray links are not explicitly modeled.

action results in no movement of the robot. However, it is clear that there exists a policy that satisfies ϕ with probability 1, thus this example satisfies the conditions for Theorem 1.

B. Control of a Traffic Network with Two Intersections

To demonstrate the utility of our approach, we apply our control synthesis algorithm to a traffic network with two signalized intersections as depicted in Fig. 2. We employ a traffic flow model with a time step of 15 seconds. At each discrete time step, signal v_1 either actuates link 1 or link 3, and signal v_2 actuates link 2 or link 4. For $i = 1, 2$, the Boolean variable s_{v_i} is equal to 1 if link i is actuated at signal v_i and is equal to 0 otherwise. The set of control actions is then $A \triangleq \{(1,2),(1,4),(3,2),(3,4)\}$ where, for $a \in A$, $l \in a$ implies that link l is actuated. The gray links in Fig. 2 are not explicitly considered in the model as they carry traffic out of the network.

The model considers a queue of vehicles waiting on each link, and at each time step, the queue is forwarded to downstream links if the queue's link is actuated and if there is available road space downstream. If the queue is longer than some *saturating limit*, then only this limit is forwarded and the remainder remains enqueue for the next time step. The vehicles that are forwarded divide among downstream links via *turn ratios* given with the model.

Let $C_l > 0$ be the capacity of link l. Here, the queue length is assumed to take on continuous values. To obtain a discrete model, the interval $[0, C_l] \subset \mathbb{R}$ is divided into a finite, disjoint set of subintervals. For example, if link l can accomodate up to $C_l = 40$ vehicles, we may divide $[0, 40]$ into the set $\{[0, 10], (10, 20], (20, 30], (30, 40]\}$. The current discrete state of link l is then the subinterval that contains the current queue length of link l, and the total state of the network is the collection of current subintervals containing the current queue lengths of each link.

Here, we consider probabilistic transitions among the discrete states and obtain an MDP model with control actions A as defined previously. For the example in Fig. 2, we have

$$(C_1, C_2, C_3, C_4) = (40, 50, 30, 30) \tag{8}$$

and link 1 is divided into four subintervals, link 2 is divided into five subintervals, and links 3 and 4 are divided into two subintervals each. In addition, we augment the state space with the last applied control action so that the control objective, expressed as a LTL formula, may include conditions on the traffic lights as is the case below, thus there are 320 total discrete states. The transition probabilities for the MDP model are determined by the specific subintervals, saturating limits, and turn ratios. Future research will investigate the details of abstracting the traffic dynamics to an MDP.

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

$$\mathbf{FG}(x_1 \leq 30 \wedge x_2 \leq 30) \wedge \tag{9}$$

$$\mathbf{GF}(x_3 \leq 10) \wedge \mathbf{GF}(x_4 \leq 10) \wedge \tag{10}$$

$$\mathbf{G}((s_{v_2} \wedge \mathbf{X}(\neg s_{v_2})) \implies (\mathbf{XX}(\neg s_{v_2}) \wedge \mathbf{XXX}(\neg s_{v_2}))). \tag{11}$$

In words, (9)–(11) 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 (11) 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 (9)–(11), 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 ϕ since x_2 remains above 30 vehicles. If estimates of turn ratios and saturation limits are available from, *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 product MDP. When applied to the true traffic model, the controller greatly outperforms the naive policy but still does not satisfy ϕ, as shown in Fig. 3(b). However, by modifying this policy via reinforcement learning on the true traffic dynamics, we obtain a controller that empirically often satisfies ϕ as seen in Fig. 3(c) (Note that we should not expect ϕ 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

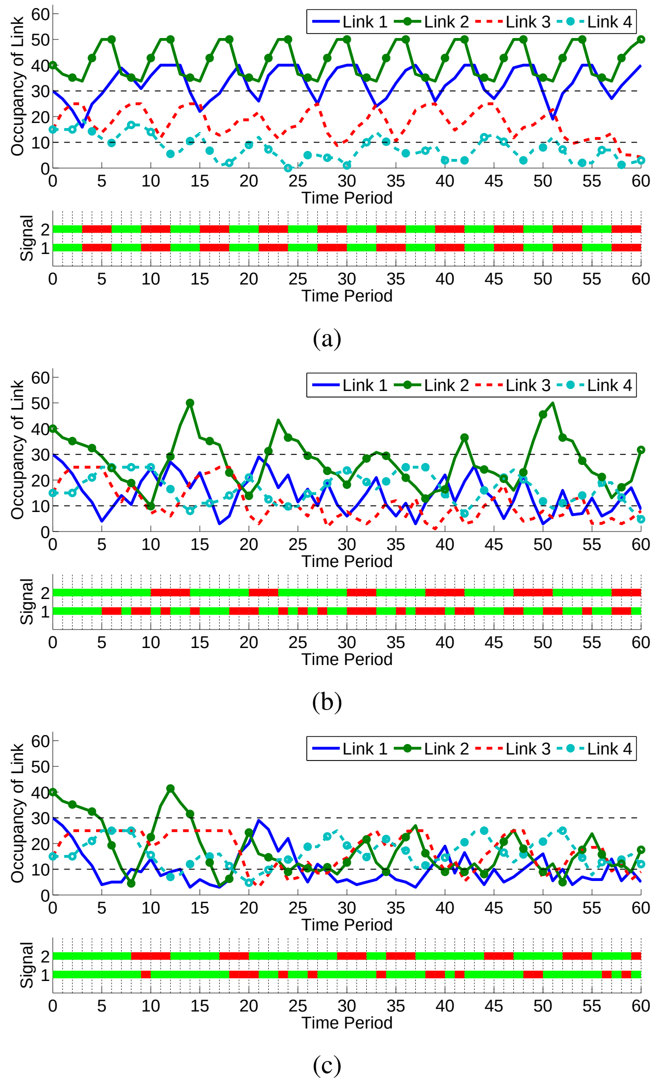

Fig. 3. Sample trajectories of the traffic network in Fig. 2. **(a)** A simple controller that synchronously actuates links for 3 time periods and does not satisfy ϕ. **(b)** An optimal controller for an MDP obtained from an approximate model of the traffic dynamics (*e.g.*, a model with turn ratios and saturation limits different than reality). This controller outperforms the previous naïve controller, but does not fully satisfy ϕ. **(c)** The controller from (b) is modified via reinforcement learning on the true traffic model. In the lower plot for all cases, signal i for $i = 1, 2$ is green if link i is actuated and is red otherwise. This example suggests how a reasonable control policy can be obtained from an approximate MDP estimated via, *e.g.*, historical data and modified "online" using reinforcement learning on observed traffic dynamics.

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.

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 (CDC), 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 markov decision processes with linear temporal logic constraints," *IEEE Transactions on Automatic Control*, vol. PP, no. 99, pp. 1–1, 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 ω-automata for formulas of linear temporal logic," in *Implementation and Application of Automata*. Springer, 2004.

[12] S. Safra, "On the complexity of ω-automata," in *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, ser. SFCS '88, 1988, pp. 319–327.

[13] D. Sadigh, E. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia, "A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-166, Sep 2014. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-166.html

[14] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[15] R. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[16] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine Learning*, no. 49, pp. 209–232, 2002.