Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications

Wongpiromsarn, Tichakorn, and Emilio Frazzoli. “Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications.” In 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), 7644-7651. Institute of Electrical and Electronics Engineers, 2012.

As Published http://dx.doi.org/10.1109/CDC.2012.6426524

Publisher Institute of Electrical and Electronics Engineers (IEEE)

Version Original manuscript

Citable link http://hdl.handle.net/1721.1/81775

Terms of Use Creative Commons Attribution-Noncommercial-Share Alike 3.0

Detailed Terms http://creativecommons.org/licenses/by-nc-sa/3.0/
Control of Probabilistic Systems under Dynamic, Partially Known Environments with Temporal Logic Specifications

Tichakorn Wongpiromsarn and Emilio Frazzoli

Abstract—We consider the synthesis of control policies for probabilistic systems, modeled by Markov decision processes, operating in partially known environments with temporal logic specifications. The environment is modeled by a set of Markov chains. Each Markov chain describes the behavior of the environment in each mode. The mode of the environment, however, is not known to the system. Two control objectives are considered: maximizing the expected probability and maximizing the worst-case probability that the system satisfies a given specification.

I. INTRODUCTION

In many applications, control systems need to perform complex tasks and interact with their (potentially adversarial) environments. The correctness of these systems typically depends on the behaviors of the environments. For example, whether an autonomous vehicle exhibits a correct behavior at a pedestrian crossing depends on the behavior of the pedestrians, e.g., whether they actually cross the road, remain on the same side of the road or step in front of the vehicle while it is moving.

Temporal logics, which were primarily developed by the formal methods community for specifying and verifying correctness of software and hardware systems, have been recently employed to express complex behaviors of control systems. Its expressive power offers extensions to properties that can be expressed than safety and stability, typically studied in the control and hybrid systems domains. In particular, [1] shows that the traffic rule enforced in the 2007 DARPA Urban Challenge can be precisely described using these logics. Furthermore, the recent development of language equivalence and simulation notions allows abstraction of continuous systems to a purely discrete model [2], [3], [4]. This subsequently provides a framework for integrating methodologies from the formal methods and the control-theoretic communities and enables formal specification, design and verification of control systems with complex behaviors.

Controller synthesis from temporal logic specifications has been considered in [5], [6], [7], [8], assuming static environments. Synthesis of reactive controllers that takes into account all the possible behaviors of dynamic environments can be found in [9], [10]. In this case, the environment is treated as an adversary and the synthesis problem can be viewed as a two-player game between the system and the environment: the environment attempts to falsify the specification while the system attempts to satisfy it [11]. In these works, the system is assumed to be deterministic, i.e., an available control action in each state enables exactly one transition. Controller synthesis for probabilistic systems such as Markov decision processes (MDP) has been considered in [12], [13]. However, these works assume that at any time instance, the state of the system, including the environment, as well as their models are fully known. This may not be a valid assumption in many applications. For example, in the pedestrian crossing problem previously described, the behavior of the pedestrians depend on their destination, which is typically not known to the system. Having to account for all the possible behaviors of the pedestrians with respect to all their possible destinations may lead to conservative results and in many cases, unrealizable specifications.

Partially observable Markov decision process (POMDP) provides a principled mathematical framework to cope with partial observability in stochastic domains [14]. Roughly, the main idea is to maintain a belief, which is defined as a probability distribution over all the possible states. POMDP algorithms then operate in the belief space. Unfortunately, it has been shown that solving POMDPs exactly is computationally intractable [15]. Hence, point-based algorithms have been developed to compute an approximate solution based on the computation over a representative set of points from the belief space rather than the entire belief space [16].

In this paper, we take an initial step towards solving POMDPs that are subject to temporal logic specifications. In particular, we consider the problem where a collection of possible environment models is available to the system. Different models correspond to different modes of the environment. However, the system does not know in which mode the environment is. In addition, the environment may change its mode during an execution subject to certain constraints.

We consider two control objectives: maximizing the expected probability and maximizing the worst-case probability that the system satisfies a given temporal logic specification. The first objective is closely related to solving POMDPs as previously described whereas the second objective is closely related to solving uncertain MDPs [17]. However, for both problems, we aim at maximizing the expected or worst-case probability of satisfying a temporal logic specification, instead of maximizing the expected or worst-case reward as considered in the POMDP and MDP literature.

The main contribution of this paper is twofold. First, we show that the expectation-based synthesis problem can be formulated as a control policy synthesis problem for MDPs under temporal logic specifications and provide a complete
solution to the problem. Second, we define a mathematical object called adversarial Markov decision process (AMDP) and show that the worst-case-based synthesis problem can be formulated as a control policy synthesis problem for AMDP. A complete solution to the control policy synthesis for AMDP is then provided. Finally, we show that the maximum worst-case probability that a given specification is satisfied does not depend on whether the controller and the adversary play alternatively or both the control and adversarial policies are computed at the beginning of an execution.

The rest of the paper is organized as follows: We provide useful definitions and descriptions of the formalisms in the following section. Section II is dedicated to the problem formulation. The expectation-based and the worst-case-based control policy synthesis are considered in Section IV and Section V, respectively. Section VI presents an example. Finally, Section VII concludes the paper and discusses future work.

II. PRELIMINARIES

We consider systems that comprise stochastic components. In this section, we define the formalisms used in this paper to describe such systems and their desired properties. Throughout the paper, we let $X^*$, $X^\omega$ and $X^+$ denote the set of finite, infinite and nonempty finite strings, respectively, of a set $X$.

A. Automata

Definition 1: A deterministic Rabin automaton (DRA) is a tuple $A = (Q, \Sigma, \delta, q_{init}, Acc)$ where

- $Q$ is a finite set of states,
- $\Sigma$ is a finite set called alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is a transition function,
- $q_{init} \in Q$ is the initial state, and
- $Acc \subseteq 2^Q \times 2^Q$ is the acceptance condition.

We use the relation notation, $q \xrightarrow{w} q'$ to denote $\delta(q, w) = q'$.

Consider an infinite string $\sigma = \sigma_0 \sigma_1 \ldots \in \Sigma^\omega$. A run for $\sigma$ in a DRA $A = (Q, \Sigma, \delta, q_{init}, Acc)$ is an infinite sequence of states $q_0 q_1 \ldots q_n$ such that $q_0 = q_{init}$ and $q_i \xrightarrow{s_i} q_{i+1}$ for all $i \geq 0$. A run is accepting if there exists a pair $(H, K) \in Acc$ such that (1) there exists $n \geq 0$ such that for all $m \geq n$, $q_m \notin H$, and (2) there exist infinitely many $n \geq 0$ such that $q_n \in K$.

A string $\sigma \in \Sigma^*$ is accepted by $A$ if there is an accepting run of $\sigma$ in $A$. The language accepted by $A$, denoted by $L_\omega(A)$, is the set of all accepted strings of $A$.

B. Linear Temporal Logic

Linear temporal logic (LTL) is a branch of logic that can be used to reason about a time line. An LTL formula is built up from a set $\Pi$ of atomic propositions, the logic connectives $\neg$, $\lor$, $\land$ and $\implies$ and the temporal modal operators $\Box$ (“next”), $\Diamond$ (“always”), $\therefore$ (“eventually”) and $U$ (“until”). An LTL formula over a set $\Pi$ of atomic propositions is inductively defined as

$$\varphi := \text{True} \mid p \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi$$

where $p \in \Pi$. Other operators can be defined as follows:

$\varphi \land \psi = \neg(\neg\varphi \lor \neg\psi)$,
$\varphi \lor \psi = \neg(\neg\varphi \land \neg\psi)$,
$\varphi = \text{True} \implies \varphi$.

and $\Box \varphi = \neg\Diamond \neg\varphi$.

Semantics of LTL: LTL formulas are interpreted on infinite strings over $2^\Pi$. Let $\sigma = \sigma_0 \sigma_1 \ldots$ where $\sigma_i \in 2^\Pi$ for all $i \geq 0$. The satisfaction relation $\models$ is defined inductively on LTL formulas as follows:

- $\sigma \models \text{True}$,
- for an atomic proposition $p \in \Pi$, $\sigma \models p$ if and only if $p \in \sigma_0$,
- $\sigma \models \neg \varphi$ if and only if $\sigma \not\models \varphi$,
- $\sigma \models \varphi_1 \land \varphi_2$ if and only if $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$,
- $\sigma \models \varphi \lor \varphi$ if and only if $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$,
- $\sigma \models \varphi \lor \varphi$ if and only if $\sigma \models \varphi_1 \land \varphi_2$ and for all $i$ such all $0 \leq i < j$, $\sigma_{i} \sigma_{j+1} \ldots \models \varphi_1$.

Given propositional formulas $p_1$ and $p_2$, examples of widely used LTL formulas include a safety formula $\Box p_1$ (read as “always $p_1$”), which simply asserts that property $p_1$ remains invariantly true throughout an execution and a reachability formula $\Diamond p_1$ (read as “eventually $p_1$”), which states that property $p_1$ becomes true at least once in an execution (i.e., there exists a reachable state that satisfies $p_1$). In the example presented later in this paper, we use a formula $p_1 \lor p_2$ (read as “$p_1$ until $p_2$”), which asserts that $p_1$ has to remain true until $p_2$ becomes true and there is some point in an execution where $p_2$ becomes true.

It can be shown that for any LTL formula $\varphi$ over $\Pi$, there exists a DRA $A$ with alphabet $\Sigma = 2^\Pi$ that accepts all and only words over $\Pi$ that satisfy $\varphi$, i.e., $L_\omega(A) = \{ \sigma \in (2^\Pi)^\omega \mid \sigma \models \varphi \}$. Such $A$ can be automatically constructed using existing tools [18]. We refer the reader to [19], [20], [21] for more details on LTL.

C. Systems and Control Policies

Definition 2: A (discrete-time) Markov chain (MC) is a tuple $M = (S, P, s_{init}, \Pi, L)$ where

- $S$ is a countable set of states,
- $P : S \times S \rightarrow [0, 1]$ is the transition probability function such that for any state $s \in S$, $\sum_{s' \in S} P(s, s') = 1$,
- $s_{init} \in S$ is the initial state,
- $\Pi$ is a set of atomic propositions, and
- $L : S \rightarrow 2^\Pi$ is a labeling function.

Definition 3: A Markov decision process (MDP) is a tuple $M = (S, Act, P, s_{init}, \Pi, L)$ where $S$, $s_{init}$, $\Pi$ and $L$ are defined as in MC and

- $\text{Act}$ is a finite set of actions, and
- $P : S \times \text{Act} \times S \rightarrow [0, 1]$ is the transition probability function such that for any state $s \in S$ and action $\alpha \in \text{Act}$, $\sum_{s' \in S} P(s, \alpha, s') = 1$.

An action $\alpha$ is enabled in state $s$ if and only if $\sum_{s' \in S} P(s, \alpha, s') = 1$. Let $\text{Act}(s)$ denote the set of enabled actions in $s$.

Given a complete system as the composition of all its components, we are interested in computing a control policy for the system that optimizes certain objectives. We define a control policy for a system modeled by an MDP as follows.
Definition 4: Let $\mathcal{M} = (S, \text{Act}, P, s_{\text{init}}, \Pi, L)$ be a Markov decision process. A control policy for $\mathcal{M}$ is a function $\mathcal{C} : S^+ \rightarrow \text{Act}$ such that $\mathcal{C}(s_0s_1 \ldots s_n) \in \text{Act}(s_n)$ for all $s_0s_1 \ldots s_n \in S^+$.

Let $\mathcal{M} = (S, \text{Act}, P, s_{\text{init}}, \Pi, L)$ be an MDP and $\mathcal{C} : S^+ \rightarrow \text{Act}$ be a control policy for $\mathcal{M}$. An infinite sequence $r^C_M = s_0s_1 \ldots$ on $\mathcal{M}$ generated under policy $\mathcal{C}$ is called a path on $\mathcal{M}$ if $s_0 = s_{\text{init}}$ and $P(s_i, C(s_0s_1 \ldots s_i), s_{i+1}) > 0$ for all $i$. The subsequence $s_0s_1 \ldots s_n$ where $n \geq 0$ is the prefix of length $n$ of $r^C_M$. We define $PPaths^C_M$ and $FPaths^C_M$ as the set of all infinite paths of $\mathcal{M}$ under policy $\mathcal{C}$ and their finite prefixes, respectively. For $s_0s_1 \ldots s_n \in FPaths^C_M$, we let $Paths^C_M(s_0s_1 \ldots s_n)$ denote the set of all paths in $PPaths^C_M$ with prefix $s_0s_1 \ldots s_n$.

The $\sigma$-algebra associated with $\mathcal{M}$ under policy $\mathcal{C}$ is defined as the smallest $\sigma$-algebra that contains $PPaths^C_M(r^C_M)$ where $r^C_M$ ranges over all finite paths in $FPaths^C_M$. It follows that there exists a unique probability measure $Pr^C_M$ on the $\sigma$-algebra associated with $\mathcal{M}$ under policy $\mathcal{C}$ where for any $s_0s_1 \ldots s_n \in FPaths^C_M$,

$$Pr^C_M(Paths^C_M(s_0s_1 \ldots s_n)) = \prod_{0 \leq i < n} P(s_i, C(s_0s_1 \ldots s_i), s_{i+1}).$$

(1)

Given an LTL formula $\varphi$, one can show that the set $(s_0s_1 \ldots s_n \in Paths^C_M) \mid L(s_0)L(s_1) \ldots = \varphi$ is measurable [19]. The probability for $\mathcal{M}$ to satisfy $\varphi$ under policy $\mathcal{C}$ is then defined as

$$Pr^C_M(\varphi) = Pr^C_M(s_0s_1 \ldots s_n \in Paths^C_M \mid L(s_0)L(s_1) \ldots = \varphi).$$

For a given (possibly noninitial) state $s \in S$, we let $\mathcal{M}^s = (S, \text{Act}, P, s, \Pi, L)$, i.e., $\mathcal{M}^s$ is the same as $\mathcal{M}$ except that its initial state is $s$. We define $Pr^C_M(s = \varphi) = Pr^C_M(\varphi)$ as the probability for $\mathcal{M}$ to satisfy $\varphi$ under policy $\mathcal{C}$, starting from $s$.

A control policy essentially resolves all the nondeterministic choices in an MDP and induces a Markov chain $\mathcal{M}_C$ that formalizes the behavior of $\mathcal{M}$ under control policy $\mathcal{C}$ [19]. In general, $\mathcal{M}_C$ contains all the states in $S^+$ and hence may not be finite even though $\mathcal{M}$ is finite. However, for a special case where $C$ is a memoryless or a finite memory control policy, it can be shown that $\mathcal{M}_C$ can be identified with a finite MC. Roughly, a memoryless control policy always picks the action based only on the current state of $\mathcal{M}$, regardless of the path that led to that state. In contrast, a finite memory control policy also maintains its “mode” and picks the action based on its current mode and the current state of $\mathcal{M}$.

III. PROBLEM FORMULATION

Consider a system that comprises 2 components: the plant and the environment. The system can regulate the state of the plant but has no control over the state of the environment. We assume that at any time instance, the state of the plant and the environment can be precisely observed.

The plant is modeled by a finite MDP $\mathcal{M}^{pl} = (Sp^l, \text{Act}, P^{pl}, s_{\text{init}}^{pl}, \Pi^{pl}, L^{pl})$. We assume that for each $s \in Sp^l$, there is an action $\alpha \in \text{Act}$ that is enabled in state $s$. In addition, we assume that the environment can be modeled by some MC $\mathcal{M}^{env} = \{M_1^{env}, M_2^{env}, \ldots, M_N^{env}\}$ where for each $i \in \{1, \ldots, N\}, M_i^{env} = (S_i^{env}, P_i^{env}, \Pi_i^{env}, \Pi_i^{env}, L_i^{env})$ is a finite MC that represents a possible model of the environment. For the simplicity of the presentation, we assume that for all $i \in \{1, \ldots, N\}$, $S_i^{env} = S^{env}$, $\Pi_i^{env} = \Pi^{env}$, $\Pi_i^{env} = S_i^{env}$, and $L_i^{env} = L^{env}$; hence, $M_1^{env}, M_2^{env}, \ldots, M_N^{env}$ differ only in the transition probability function. These different environment models can be considered as different modes of the environment. For the rest of the paper, we use “environment model” and “environment mode” interchangeably to refer to some $M_i^{env} \in M^{env}$.

We further assume that the plant and the environment make a transition simultaneously, i.e., both of them makes a transition at every time step. All $M_i^{env} \in M^{env}$ are available to the system. However, the system does not know exactly which $M_i^{env} \in M^{env}$ is the actual model of the environment. Instead, it maintains the belief $B : M^{env} \rightarrow [0, 1]$, which is defined as a probability distribution over all possible environment models such that $\sum_{1 \leq i \leq N} B(M_i^{env}) = 1$. $B(M_i^{env})$ returns the probability that $M_i^{env}$ is the model being executed by the environment. The set of all the beliefs forms the belief space, which we denote by $B$. In order to obtain the belief at each time step, the system is given the initial belief $B_{\text{init}} : M^{env} \rightarrow [0, 1] \in B$. Then, it subsequently updates the belief using a given belief update function $\tau : B \times S^{env} \times S^{env} \rightarrow B$ such that $\tau(B, s, s')$ returns the belief after the environment makes a transition from state $s$ with belief $B$ to state $s'$. The belief update function can be defined based on the observation function as in the belief MDP construction for POMDPs [14].

In general, the belief space $B$ may be infinite, rendering the control policy synthesis computationally intractable to solve exactly. To overcome this difficulty, we employ techniques for solving POMDPs and approximate $B$ by a finite set of representative points from $B$ and work with this approximate representation instead. Belief space approximation is beyond the scope of this paper and is subject to future work. Sampling techniques that have been proposed in the POMDP literature can be found in, e.g., [16], [22], [23].

Given a system model described by $\mathcal{M}^{pl}$, $M^{env} = \{M_1^{env}, M_2^{env}, \ldots, M_N^{env}\}$, the (finite) belief space $B$, the initial belief $B_{\text{init}}$, the belief update function $\tau$ and an LTL formula $\varphi$ that describes the desired property of the system, we consider the following control policy synthesis problems.

Problem 1: Synthesize a control policy for the system that maximizes the expected probability that the system satisfies $\varphi$ where the expected probability that the environment transitions from state $s \in S^{env}$ with belief $B \in B$ to state $s' \in S^{env}$ is given by $\sum_{1 \leq i \leq N} B(M_i^{env}) P_i^{env}(s, s')$.

Problem 2: Synthesize a control policy for the system that maximizes the worst-case (among all the possible sequences of environment modes) probability that the system satisfies $\varphi$. The environment mode may change during an execution: when the environment is in state $s \in S^{env}$ with belief $B \in B$,
it may switch to any mode $M_i^{env} \in M^{env}$ with $B(M_i^{env}) > 0$. We consider both the case where the controller and the environment plays a sequential game and the case where the control policy and the sequence of environment modes are computed before an execution.

Example 1: Consider a problem where an autonomous vehicle needs to navigate a road with a pedestrian walking on the pavement. The vehicle and the pedestrian are considered the plant and the environment, respectively. The pedestrian may or may not cross the road, depending on his/her destination, which is unknown to the system. Suppose the road is discretized into a finite number of cells $c_0, c_1, \ldots, c_M$. The vehicle is modeled by an MDP $M^{pl} = (S^{pl}, Act, P^{pl}, s^{init, pl}, \Pi^{pl}, L^{pl})$ whose state $s \in S^{pl}$ describes the cell occupied by the vehicle and whose action $\alpha \in Act$ corresponds to a motion primitive of the vehicle (e.g., cruise, accelerate, decelerate). The motion of the pedestrian is modeled by a MC $M_i^{env} \in M^{env}$ where $M^{env} = \{M_1^{env}, M_2^{env}\}$. $M_1^{env} = (S^{env}, P_1^{env}, s^{init, env}, \Pi^{env}, L^{env})$ represents the model of the pedestrian if s/he decides not to cross the road whereas $M_2^{env} = (S^{env}, P_2^{env}, s^{init, env}, \Pi^{env}, L^{env})$ represents the model of the pedestrian if s/he decides to cross the road. A state $s \in S^{env}$ describes the cell occupied by the pedestrian. The labeling functions $L^{pl}$ and $L^{env}$ essentially map each cell to its label, with an index that identifies the vehicle from the pedestrian, i.e., $L^{pl}(c_j) = c_j^{pl}$ and $L^{env}(c_j) = c_j^{env}$ for all $j \in \{0, \ldots, M\}$. Consider the desired property stating that the vehicle does not collide with the pedestrian until it reaches cell $c_M$ (e.g., the end of the road). In this case, the specification $\varphi$ can be written as $\varphi = \neg \bigvee_{j \geq 0} (c_j^{pl} \wedge c_j^{env}) \cup \varphi_M^{pl}$.

IV. EXPECTATION-BASED CONTROL POLICY SYNTHESIS

To solve Problem 1, we first construct the MDP that represents the complete system, taking into account the uncertainties, captured by the belief, in the environment model. Then, we employ existing results in probabilistic uncertainties, captured by the belief, in the environment represents the complete system, taking into account the

Definition 5: Let $M_1 = (S_1, Act, P_1, s_{init, 1}, \Pi_1, L_1)$ be a Markov decision process and $M_2 = (S_2, P_2, s_{init, 2}, \Pi_2, L_2)$ be a Markov chain. Their synchronous parallel composition, denoted by $M_1 \parallel M_2$, is the MDP $M = (S_1 \times S_2, Act, P, (s_{init, 1}, s_{init, 2}), \Pi_1 \cup \Pi_2, L)$ where:

- For each $s_1, s_1' \in S_1$, $s_2, s_2' \in S_2$ and $\alpha \in Act$, $P((s_1, s_2), (s_1', s_2')) = P_1(s_1, \alpha, s_1') P_2(s_2, s_2')$.
- For each $s_1 \in S_1$ and $s_2 \in S_2$, $L((s_1, s_2)) = L(s_1)$.

From the above definitions, our complete system can be modeled by the MDP $M^{pl} \parallel M^{env}$. We denote this MDP by $M = (S, Act, P, s_{init}, \Pi, L)$. Note that a state $s \in S$ is of the form $s = (s^{pl}, s^{env}, B)$ where $s^{pl} \in S^{pl}$, $s^{env} \in S^{env}$ and $B \in B$. The following lemma shows that Problem 1 can be solved by finding a control policy $C$ for $M$ that maximizes $\Pr^C_{M}(\varphi)$.

Lemma 1: Let $r^C_{M}(s_0, s_1, \ldots, s_n) = \xi$ be a finite path of $M$ under policy $C$ for each $i, s_i = (s_i^{pl}, s_i^{env}, B_i) \in S^{pl} \times S^{env} \times B$. Then,

$$
Pr^C_{M}(\text{Path}^C_{M}(r^C_{M})) = \prod_{0 \leq j < n} (P(s_j^{pl} | C(s_0, s_1, \ldots, s_j), s_j^{pl}+1)) \sum_{1 \leq i \leq n} B_i(M_i^{env})(s_i^{env} | s_i^{pl+1} + 1)
$$

Hence, given an LTL formula $\varphi$, $Pr^C_{M}(\varphi)$ gives the expected probability that the system satisfies $\varphi$ under policy $C$.

Proof: The proof straightforwardly follows from the definition of $M^{env}$ and $M$.

B. Construction of the Product MDP

Let $A_{\varphi} = (Q, 2^Q, \delta, q_{init}, Acc)$ be a DRA that recognizes the specification $\varphi$. Our next step is to obtain a finite MDP $M_p = (S_p, Act_p, P_p, s_{init, p}, \Pi_p, L_p)$ as the product of $M$ and $A_{\varphi}$, defined as follows.

Definition 6: Let $M = (S, Act, P, s_{init}, \Pi, L)$ be an MDP and let $A = (Q, 2^Q, \delta, q_{init}, Acc)$ be a DRA. Then, the product of $M$ and $A$ is the MDP $M_p = M \otimes A$ defined by $M_p = (S_p, Act_p, P_p, s_{init, p}, \Pi_p, L_p)$ where $S_p = S \times Q$, $s_{init, p} = (s_{init}, q_{init}, L(s_{init}))$, $\Pi_p = Q$, $L_p(s, q) = \{q\}$ and

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

Consider a path $r^C_{M_p}(s_0, q_0, s_1, q_1, \ldots, s_n, q_n) \ldots$ of $M_p$ under some control policy $C_p$. We say that $r^C_{M_p}$ is accepting if and only if there exists a pair $(H, K) \in Acc$ such that the word generated by $r^C_{M_p}$ intersects with $H$ finitely many times and intersects with $K$ infinitely many times, i.e., (1) there exists $n \geq 0$ such that for all $m \geq n$, $L_p((s_m, q_m)) \cap H = \emptyset$, and (2) there exists infinitely many $n \geq 0$ such that $L_p((s_n, q_n)) \cap K \neq \emptyset$. 

Stepping through the above definition shows that given a path $r_{M_p}^C = (s_0, q_0) (s_1, q_1) \ldots$ of $M_p$ generated under some control policy $C_p$, the corresponding path $s_0, s_1 \ldots$ on $\mathcal{M}$ generates a word $L(s_0) L(s_1) \ldots$ that satisfies $\varphi$ if and only if $r_{M_p}^C$ is accepting. Therefore, each accepting path of $M_p$ uniquely corresponds to a path of $\mathcal{M}$ whose word satisfies $\varphi$. In addition, a control policy $C_p$ on $M_p$ induces a corresponding control policy $C$ on $\mathcal{M}$. The details for generating $C$ from $C_p$ can be found, e.g., in [19], [12].

C. Control Policy Synthesis for Product MDP

From probabilistic verification, it has been shown that the maximum probability for $\mathcal{M}$ to satisfy $\varphi$ is equivalent to the maximum probability of reaching a certain set of states of $M_p$ known as accepting maximal end components (AMECs). An end component of the product MDP $M_p = (S_p, Act_p, P_p, s_p, \Pi_p, L_p)$ is a pair $(T, A)$ where $T \neq S$ and $A : T \rightarrow 2^{Act_p}$ such that (1) $\emptyset \neq A(s) \subseteq Act_p(s)$ for all $s \in T$, (2) the directed graph induced by $(T, A)$ is strongly connected, and (3) for all $s \in T$ and $\alpha \in A(s)$, $\{t \in S_p \mid P_p(s, \alpha, t) > 0\} \subseteq T$. An accepting maximal end component of $M_p$ is an end component $(T, A)$ such that for some $(H, K) \in Acc$, $H \cap T = \emptyset$ and $K \cap T = \emptyset$ and there is no end component $(T', A')$ (\(T', A') \neq (T, A)$) such that $T \subseteq T'$ and $A(s) \subseteq A'(s)$ for all $s \in T$. It has an important property that starting from any state in $T$, there exists a finite memory control policy to keep the state within $T$ forever while visiting all states in $T$ infinitely often with probability 1. AMECs of $M_p$ can be efficiently identified based on iterative computations of strongly connected components of $M_p$. We refer the reader to [19] for more details.

Once the AMECs of $M_p$ are identified, we then compute the maximum probability of reaching $S_G$ where $S_G$ contains all the states in the AMECs of $M_p$. For the rest of the paper, we use an LTL-like notation to describe events in MDPs. In particular, we use $\diamond S_G$ to denote the event of reaching some state in $S_G$ eventually.

For each $s \in S_p$, let $x_s$ denote the maximum probability of reaching a state in $S_G$, starting from $s$. Formally, $x_s = \sup_{C_p} \max_{s \in G} P(s, \alpha, t) x_t$ for all $s \in S_p$. There are two main techniques for computing the probability $x_s$ for each $s \in S_p$: linear programming (LP) and value iteration. LP-based techniques yield an exact solution but it typically does not scale as well as value iteration. On the other hand, value iteration is an iterative numerical technique. This method works by successively computing the probability vector $(x_s(k))_{s \in S_p}$ for increasing $k \geq 0$ such that $\lim_{k \rightarrow \infty} x_s(k) = x_s$ for all $s \in S_p$. Initially, we set $x_s(0) = 1$ if $s \in S_G$ and $x_s(0) = 0$ otherwise. In the $(k+1)$th iteration where $k \geq 0$, we set

$$x_s^{(k+1)} = \begin{cases} 1 & \text{if } s \in S_G \\ \max_{\alpha \in Act_p(s)} \sum_{t \in S_p} P(s, \alpha, t) x_t^{(k)} & \text{otherwise}. \end{cases}$$

In practice, we terminate the computation and say that $x_s^{(k)}$ converges when a termination criterion such as $\max_{s \in S_p} |x_s^{(k+1)} - x_s^{(k)}| < \epsilon$ is satisfied for some fixed (typically very small) threshold $\epsilon$.

Once the vector $(x_s)_{s \in S_p}$ is computed, a finite memory control policy $C_p$ for $M_p$ that maximizes the probability for $\mathcal{M}$ to satisfy $\varphi$ can be constructed as follows. First, consider the case when $M_p$ is in state $s \in S_G$. In this case, $s$ belongs to some AMEC $(T, A)$ and the policy $C_p$ selects an action $\alpha \in A(s)$ such that all actions in $A(s)$ are scheduled infinitely often. (For example, $C_p$ may select the action for $s$ according to a round-robin policy.) Next, consider the case when $M_p$ is in state $s \in S_p \setminus S_G$. In this case, $C_p$ picks an action to ensure that $P(C_p) \in S_G = x_s$ can be achieved. If $x_s = 0$, an action in $Act_p(s)$ can be chosen arbitrarily. Otherwise, $C_p$ picks an action $\alpha \in Act_p^{max}(s)$ such that $P_p(s, \alpha, t) > 0$ for some $t \in S_p$ with $\|t\| = \|s\| - 1$. Here, $Act_p^{max}(s) \subseteq Act_p(s)$ is the set of actions such that for all $\alpha \in Act_p^{max}(s)$, $x_s = \sum_{t \in S_p} P(s, \alpha, t) x_t$ and $\|s\|$ denotes the length of a shortest path from $s$ to a state in $S_G$, using only actions in $Act_p^{max}$.

V. WORST-CASE-BASED CONTROL POLICY SYNTHESIS

To solve Problem 2, we first propose a mathematical object called adversarial Markov decision process (AMDP). Then, we show that Problem 2 can be formulated as finding an optimal control policy for an AMDP. Finally, control policy synthesis for AMDP is discussed.

A. Adversarial Markov Decision Process

Definition 7: An adversarial Markov decision process (AMDP) is a tuple $\mathcal{M}^A = (S, Act_C, Act_A, P, s_{init}, \Pi, L)$ where $S, s_{init}, \Pi$ and $L$ are defined as in MDP and

- $Act_C$ is a finite set of control actions,
- $Act_A$ is a finite set of adversarial actions, and
- $P : S \times Act_C \times Act_A \times S \rightarrow [0, 1]$ is the transition probability function such that for any $s \in S, \alpha \in Act_C$ and $\beta \in Act_A, \sum_{s' \in S} P(s, \alpha, \beta, s') = 1$.

We say that a control action $\alpha$ is enabled in state $s$ if and only if there exists an adversarial action $\beta$ such that $\sum_{s' \in S} P(s, \alpha, \beta, s') = 1$. Similarly, an adversarial action $\beta$ is enabled in state $s$ if and only if there exists a control action $\alpha$ such that $\sum_{s' \in S} P(s, \alpha, \beta, s') = 1$. Let $Act_C(s)$ and $Act_A(s)$ denote the set of enabled control and adversarial actions, respectively, in $s$. We assume that for all $s \in S, \alpha \in Act_C(s)$ and $\beta \in Act_A(s), \sum_{s' \in S} P(s, \alpha, \beta, s') = 1$, i.e., whether an adversarial (resp. control) action is enabled in state $s$ depends only on the state $s$ itself but not on a control (resp. adversarial) action taken by the system (resp. adversary).

Given an AMDP $\mathcal{M}^A = (S, Act_C, Act_A, P, s_{init}, \Pi, L)$, a control policy $\mathcal{C} : S^+ \rightarrow Act_C$ and an adversarial policy $\mathcal{D} : S^+ \rightarrow Act_A$ for an AMDP can be defined such that $\mathcal{C}(s_0, s_1, \ldots, s_n) \in Act_C(s_n)$ and $\mathcal{D}(s_0, s_1, \ldots, s_n) \in Act_A(s_n)$ for all $s_0, s_1, \ldots, s_n \in S^+$. A unique policy measure $P_{\mathcal{M}^A}^{\mathcal{C}, \mathcal{D}}$ on the $\sigma$-algebra associated with $\mathcal{M}^A$ under control policy $\mathcal{C}$ and adversarial policy $\mathcal{D}$ can then be defined based on the notion of path on $\mathcal{M}^A$ as for an ordinary MDP.
We end the section with important properties of AMDP that will be employed in the control policy synthesis.

**Definition 8:** Let $G$ be a set of functions from $T$ to $\mathcal{V}$ where $T$ and $\mathcal{V}$ are finite sets. We say that $G$ is complete if for any $t_1, t_2 \in T$ and $g_1, g_2 \in G$, there exists $g \in G$ such that $g(t_1) = g_1(t_1)$ and $g(t_2) = g_2(t_2)$.

**Lemma 2:** Let $T$, $\mathcal{U}$ and $\mathcal{V}$ be finite sets and let $G$ be a set of functions from $T$ to $\mathcal{V}$. Then, $G$ is finite. Furthermore, suppose that $G$ is complete. Then, for any $F : \mathcal{U} \times T \rightarrow \mathbb{R}_{\geq 0}$ and $G : \mathcal{V} \times T \rightarrow \mathbb{R}_{\geq 0}$,

$$\min_{u \in \mathcal{U}} \sum_{t \in T} F(u, t) \max_{g \in G} G(g(t), t) = \min_{u \in \mathcal{U}} \max_{g \in G} \sum_{t \in T} F(u, t) G(g(t), t)$$

In addition,

$$\min_{u \in \mathcal{U}} \sum_{t \in T} F(u, t) \min_{g \in G} G(g(t), t) = \min_{u \in \mathcal{U}} \min_{g \in G} \sum_{t \in T} F(u, t) G(g(t), t).$$

**Proof:** Since both $T$ and $\mathcal{V}$ are finite, clearly, $G$ is finite. Thus, the $\min$ and $\max$ in (6) are well defined. Let $\bar{g} \in \bar{G}$ be a function such that $\sum_{t \in T} F(u, t) G(\bar{g}(t), t) = \max_{g \in G} \sum_{t \in T} F(u, t) G(g(t), t)$. Since $G$ is complete, there exists a function $g^* \in G$ such that for all $t \in T$, $g^*(t)$ satisfies $G(g^*(t), t) = \max_{g \in G} G(g(t), t)$. Furthermore, since both $F$ and $G$ are non-negative and $\bar{g} \in \bar{G}$, it follows that $F(u, t) G(g^*(t), t) = \max_{g \in G} F(u, t) G(g(t), t)$ for all $u \in \mathcal{U}$ and $t \in T$. Taking the sum over all $t \in T$, we get

$$\sum_{t \in T} F(u, t) G(g^*(t), t) = \sum_{t \in T} F(u, t) G(\bar{g}(t), t)$$

But since $g^* \in G$, it follows that

$$\max_{g \in G} \sum_{t \in T} F(u, t) G(g(t), t) \geq \sum_{t \in T} F(u, t) G(g^*(t), t)$$

Combining (8) and (9), we get that all the inequalities must be replaced by equalities. Hence, we can conclude that the first equality in (6) holds. The proof for the equality in (7) follows similar arguments. Hence, we only provide a proof for the second inequality in (6).

First, from weak duality, we know that

$$\min_{u \in \mathcal{U}} \max_{g \in G} \sum_{t \in T} F(u, t) G(g(t), t) \geq \sum_{t \in T} F(u^*, t) G(g^*(t), t)$$

For each $g \in G$, consider an element $u^*_g \in \mathcal{U}$ such that $\sum_{t \in T} F(u^*_g, t) G(g(t), t) = \min_{u \in \mathcal{U}} \sum_{t \in T} F(u, t) G(g(t), t)$. Since $g^* \in G$, it follows that

$$\max_{g \in G} \sum_{u \in \mathcal{U}} F(u, t) G(g(t), t) \geq \sum_{t \in T} F(u^*_g, t) G(g^*(t), t).$$

But, from the definition of $g^*$ and $u^*$ and the first equality in (6), we also get

$$\sum_{t \in T} F(u^*_g, t) G(g^*(t), t) = \min_{u \in \mathcal{U}} \sum_{t \in T} F(u, t) \max_{g \in G} G(g(t), t) = \min_{u \in \mathcal{U}} \max_{g \in G} \sum_{t \in T} F(u, t) G(g(t), t).$$

Following the chain of inequalities in (10)-(12), we can conclude that the second equality in (6) holds.

**Proposition 1:** Let $\mathcal{M}^A = (S, Act_C, Act_A, \mathbb{P}, s_{init}, \mathbb{P})$ be a finite AMDP and $S_G \subseteq S$ be the set of goal states. Let

$$x_s = \sup_{c_0 \in C_0} \inf_{D_0 \in D_0} \sup_{c_1 \in C_1} \inf_{D_1 \in D_1} \ldots \sup_{c_{k-1} \in C_{k-1}} \inf_{D_{k-1} \in D_{k-1}} \mathbb{P}^C, D_t (s \models \Diamond S_G),$$

(13)

where for any $n \geq 0$, $C_n = \{ C : S^{n+1} \rightarrow Act_C | (s_0 \ldots s_n) \in Act_C(s_0), D_n = \{ D : S^{n+1} \rightarrow Act_A | (s_0 \ldots s_n) \in Act_A(s_0), C_0(s_0 \ldots s_n) = C_n(s_0 \ldots s_n) \}$ and $D(s_0 \ldots s_n) = D_n(s_0 \ldots s_n)$. For each $k \geq 0$, consider a vector $(x_s^{(k)})_{s \in S}$ where $x_s^{(0)} = 1$ for all $s \in S_G$, $x_s^{(0)} = 0$ for all $s \notin S_G$ and for all $k \geq 0$,

$$x_s^{(k+1)} = \begin{cases} \max_{\alpha \in Act_C(s)} \min_{\beta \in Act_A(s)} \sum_{t \in S} \mathbb{P}(s, \alpha, \beta, t) x_t^{(k)} & \text{if } s \in S_G, \\ 0 & \text{otherwise} \end{cases}$$

(14)

Then, for any $s$ in $S$, $x_s^{(0)} \leq x_s^{(1)} \leq \ldots \leq x_s$ and $x_s = \lim_{k \to \infty} x_s^{(k)}$.

**Proof:** Since $\sum_{s \in S} \mathbb{P}(s, \alpha, \beta, t) \in \{0, 1\}$ for all $\alpha \in Act_C$ and $\beta \in Act_A$, it can be checked that for any $k \geq 0$, if $x_s^{(k)} \notin [0, 1]$ for all $s \in S$, then $x_s^{(k+1)} \notin [0, 1]$ for all $s \in S$. Since, $x_s^{(0)} \notin [0, 1]$ for all $s \in S$, we can conclude that $x_s^{(k)} \notin [0, 1]$ for all $k \geq 0$ and $s \in S$. Let $S_G^{\leq k}$ denote the event of reaching some state in $S_G$ within $k$ steps. We will show, using induction on $k$, that for any $k \geq 0$ and $s \in S$, $x_s^{(k)} = \sup_{c_0 \in C_0} \inf_{D_0 \in D_0} \sup_{c_1 \in C_1} \inf_{D_1 \in D_1} \ldots \sup_{c_{k-1} \in C_{k-1}} \inf_{D_{k-1} \in D_{k-1}} \mathbb{P}^C, D_t (s \models \Diamond S_G),$

(15)

where $C$ and $D$ are control and adversarial policies such that for any $n$ such that $0 \leq n < k$, $C_0(s_0 \ldots s_n) = C_n(s_0 \ldots s_n)$ and $D(s_0 \ldots s_n) = D_n(s_0 \ldots s_n)$. The case where $s \in S_G$ is trivial so we only consider an arbitrary $s \in S \setminus S_G$. Clearly, $x_s^{(0)} = \Pr_{\mathcal{M}_A}^C(s \models \Diamond S_G)$ for any control and adversarial policies $C$ and $D$. Consider an arbitrary $k \geq 0$ and assume that for all $s \in S \setminus S_G$, (15) holds. Then, from (14), we get that for any $s \in S \setminus S_G$,

$$x_s^{(k+1)} = \max_{\alpha \in Act_C(s)} \min_{\beta \in Act_A(s)} \sum_{t \in S} \mathbb{P}(s, \alpha, \beta, t) x_t^{(k)}$$

Since for all $n$ such that $0 \leq n < k$, $S^{n+1}, Act_C$ and $Act_A$ are finite, it follows that $C_n$ and $D_n$ are finite. Furthermore, $\Pr_{\mathcal{M}_A}^C(t \models \Diamond S_G)$ only depends on $C_0, D_0, \ldots, C_{k-1}$. $D_{k-1}$. Thus, we can conclude that all the sup and inf above can be attained, so we can replace them by max and min, respectively.

Consider arbitrary $s \in S \setminus S_G$ and $\alpha \in Act_C(s)$. Define a function $F : Act_A(s) \times S \rightarrow [0, 1]$ such that $F(\beta, t) = \mathbb{P}(s, \alpha, \beta, t)$. In addition, define a set $G = C_0$ and a function $G : Act_C \times S \rightarrow [0, 1]$ such that $G(C_0(t), t) = \inf_{D_0 \in D_0} \sup_{C_1 \in C_1} \inf_{D_1 \in D_1} \ldots \sup_{C_{k-1} \in C_{k-1}} \inf_{D_{k-1} \in D_{k-1}} \Pr^C, D_t (s \models \Diamond S_G)$ where for any $n \in \{0, \ldots, k-1\}$,
$C(s_0 s_1 \ldots s_n) = C_0(s_0 s_1 \ldots s_n)$ and $D(s_0 s_1 \ldots s_n) = D_0(s_0 s_1 \ldots s_n)$. Pick arbitrary $t_1, t_2 \in S$ and $g_1, g_2 \in \mathbb{G}$. Suppose $g_1(t_1) = \alpha_1$ and $g_2(t_2) = \alpha_2$. Then, from the definition of $C_0$, $\alpha_1 \in Act_C(t_1)$ and $\alpha_2 \in Act_C(t_2)$; hence, there must exists $g \in \mathbb{G}$ such that $g(t_1) = \alpha_1$ and $g(t_2) = \alpha_2$. Thus, by definition, $\mathbb{G}$ is complete. Applying Lemma 2 we get

$$x_s^{(k+1)} = \max_{\alpha \in Act_C(s)} \min_{\beta \in Act_A(s)} \max_{C_0 \in C_0} \sum_{t \in S} P(s, \alpha, \beta, t) \min_{D_0 \in D_0} \cdots \min_{c_{k-1} \in C_{k-1}} \min_{d_{k-1} \in D_{k-1}} \Pr_{C,D} \left( t = \cdot \right) \Delta \leq c S_G.$$}

Applying a similar procedure as in the previous paragraph $2k$ times, we get

$$x_s^{(k+1)} = \max_{\alpha \in Act_C(s)} \min_{\beta \in Act_A(s)} \max_{C_0 \in C_0} \sum_{t \in S} P(s, \alpha, \beta, t) \min_{D_0 \in D_0} \cdots \min_{c_{k-1} \in C_{k-1}} \min_{d_{k-1} \in D_{k-1}} \Pr_{C,D} \left( t = \cdot \right) \Delta \leq c S_G.$$

Thus, we can conclude that for any $k \geq 0$ and $s \in S \backslash S_G$, (15) holds. Furthermore, since the set of events $\Delta \leq c S_G$ includes the set of events $\Delta \leq c S_G$, we obtain $x_s^{(k)} \leq x_s^{(k+1)}$. Finally, we can conclude that $\lim_{k \to \infty} x_s^{(k)} = x_s$ using the fact that the sequence $x_s^{(0)}, x_s^{(1)}, \ldots$ is monotonic and bounded (and hence has a finite limit) and $\Delta S_G$ is the countable union of the events $\Delta \leq c S_G$.

**Proposition 2:** Let $\mathcal{M}_A = (S, Act_C, Act_A, P, s_{init}, \Pi, L)$ be a finite AMDP and $S_G \subseteq S$ be the set of goal states. Let

$$y_s = \sup_{C} \inf_{D} \Pr_{C,D} \left( s = \cdot \right) \Delta S_G. \quad (16)$$

For each $k \geq 0$, consider a vector $(y_s^{(k)})_{s \in S}$ where $y_s^{(k)} = 1$ for all $s \in S_G$, $y_s^{(0)} = 0$ for all $s \not\in S_G$ and for all $k \geq 0$,

$$y_s^{(k+1)} = \left\{ \begin{array}{ll} 1 & \text{if } s \in S_G \\
\max_{\alpha \in Act_C(s)} \min_{\beta \in Act_A(s)} \sum_{t \in S} P(s, \alpha, \beta, t) y_t^{(k)} & \text{otherwise} \end{array} \right. \quad (17)$$

Then, for any $s \in S$, $y_s^{(0)} \leq y_s^{(1)} \leq \ldots \leq y_s$ and $y_s = \lim_{k \to \infty} y_s^{(k)}$.

**Proof:** The proof closely follows the proof of Proposition 1. Roughly, we show, by induction on $k$ and applying Lemma 2 that for any $k \geq 0$ and $s \in S$, $y_s^{(k)} = \sup_{C} \inf_{D} \Pr_{C,D} \left( s = \cdot \right) \Delta \leq c S_G$. We can conclude the proof using a similar argument as in the proof of Proposition 1.

From Proposition 1 and Proposition 2, we can conclude that the sequential game (13) is equivalent to its nonsequential counterpart (16).

**Corollary 1:** The maximum worst-case probability of reaching a set $S_G$ of states in an AMDP $\mathcal{M}_A$ does not depend on whether the controller and the adversary play alternatively or both the control and adversarial policies are computed at the beginning of an execution.

**B. The Complete System as an AMDP**

We start by constructing an AMDP $\mathcal{M} = (S, Act, P, s_{init}, \Pi, L)$ that represents the complete system as described in Section IV-A. As discussed earlier, a state of $\mathcal{M}$ is of the form $(s^{pl}, s^{env}, B)$ where $s^{pl} \in S^{pl}$, $s^{env} \in S^{env}$ and $B \in \mathbb{B}$. The corresponding AMDP $\mathcal{M}_A$ of $\mathcal{M}$ is then defined as $\mathcal{M}_A = (S, Act_C, Act_A, P_A, s_{init}, \Pi, L)$ where $Act_C = Act, Act_A = \{\beta_1, \ldots, \beta_N\}$ (i.e., $\beta$ corresponds to the environment choosing model $M^{env}$) and for any $s_1^{pl}, s_2^{pl} \in S^{pl}$, $s_1^{env}, s_2^{env} \in S^{env}$, $B_1, B_2 \in \mathbb{B}$, $\alpha \in Act_C$ and $1 \leq i \leq N$,

$$P_A((s_1^{pl}, s_1^{env}, B_1), \alpha, \beta_i, (s_2^{pl}, s_2^{env}, B_2)) = \left\{ \begin{array}{ll} P^{pl}(s_1^{pl}, \alpha, s_1^{pl}) P^{env}(s_1^{env}, s_2^{env}) & \text{if } B_1(M_i^{env}) > 0 \\
0 & \text{otherwise} \end{array} \right. \quad (18)$$

Thus, it is straightforward to check that $\mathcal{M}_A$ is a valid AMDP. Furthermore, based on this construction and the assumptions that (1) at any plant state $s^{pl} \in S^{pl}$, there exists an action that is enabled in $s^{pl}$, and (2) at any point in an execution, the belief $B \in \mathbb{B}$ satisﬁes $\sum_{1 \leq i \leq N} B(M_i^{env}) = 1$, it can be shown that at any state $s \in S$, there exists a control action $\alpha \in Act_C$ and an adversarial action $\beta \in Act_A$ that are enabled in $s$. In addition, consider the case where the environment is in state $s^{env} \in S^{env}$ with belief $B \in \mathbb{B}$. It can be shown that for all $M_i^{env} \in M^{env}$, if $B(M_i^{env}) > 0$, then $\beta_i$ is enabled in $(s^{pl}, s^{env}, B)$ for all $s^{pl} \in S^{pl}$. Thus, we can conclude that $\mathcal{M}_A$ represents the complete system for Problem 2.

**Remark 1:** According to (18), the system does not need to maintain the exact belief in each state. The only information needed to construct an AMDP that represents the complete system is all the possible modes of the environment in each state of the complete system. This allows us to integrate methodologies for discrete state estimation [24] to reduce the size of the AMDP. This direction is subject to future work.

**C. Control Policy Synthesis for AMDP**

Similar to control policy synthesis for MDP, control policy synthesis for AMDP $\mathcal{M}_A$ can be done on the basis of a product construction. The product of $\mathcal{M}_A = (S, Act_C, Act_A, P, s_{init}, \Pi, L)$ and DRA $\mathcal{A} = (Q, G, \delta, q_{init}, Acc)$ is an AMDP $\mathcal{M}_A^p = (S_p, Act_C, Act_A, P_p, s_{p init}, \Pi_p, L_p)$, which is defined similar to the product of MDP and DRA, except that the set of actions is partitioned into the set of control and the set of adversarial actions.

Following the steps for synthesizing a control policy for product MDP, we identify the AMECs of $\mathcal{M}_A^p$. An AMEC of $\mathcal{M}_A^p$ is defined based on the notion of end component as for the case of product MDP. However, an end component of $\mathcal{M}_A^p$ needs to be defined, taking into account the adversary. Specifically, an end component of $\mathcal{M}_A^p$ is a pair $(T, A)$ where
Fig. 1. The road and its partition used in the autonomous vehicle example.

\( \emptyset \neq T \subseteq S_p \) and \( A : T \to 2^{Act} \) such that (1) \( \emptyset \neq A(s) \subseteq \text{Act}(s) \) for all \( s \in T \), (2) the directed graph induced by \((T, A)\) under any adversarial policy is strongly connected, and (3) for all \( s \in T \), \( \alpha \in A(s) \) and \( \beta \in \text{Act}(s) \), \( \{ t \in S_p \mid P_p(s, \alpha, \beta, t) > 0 \} \subseteq T \).

Using a similar argument as in the case of product MDP [19], it can be shown that the maximum worst-case probability for \( M^A \) to satisfy \( \varphi \) is equivalent to the maximum worst-case probability of reaching a state in an AMEC of \( M^A_p \). We can then apply Proposition 1 and Proposition 2 to compute \( x_s \), which is equivalent to \( y_s \), using value iteration. A control policy for \( M^A \) that maximizes the worst-case probability for \( M^A \) to satisfy \( \varphi \) can be constructed as outlined in the end of Section IV-C for product MDP.

VI. EXAMPLE

Consider, once again, the autonomous vehicle problem described in Example 1. Suppose the road is discretized into 9 cells \( c_0, \ldots, c_8 \) as shown in Figure 1. The vehicle starts in cell \( c_0 \) and has to reach cell \( c_8 \) whereas the pedestrian starts in cell \( c_1 \). The models of the vehicle and the pedestrian are shown in Figure 2. The vehicle has two actions \( \alpha_1 \) and \( \alpha_2 \), which correspond to decelerating and accelerating, respectively. The pedestrian has 2 modes \( M_1^{env} \) and \( M_2^{env} \), which correspond to cases where s/he wants to remain on the left side of the road and cross the road, respectively. A DRA \( A_\varphi \) that accepts all and only words that satisfy \( \varphi = (\neg \bigvee_{j \geq 0} (c_j^{pl} \land c_{j+1}^{env}) \) is shown in Figure 3. Finally, we consider the set \( B = \{ B_0, \ldots, B_6 \} \) of beliefs where for all \( i \), \( B_i(M_1^{env}) = 0.1i \) and \( B_i(M_2^{env}) = 1 - 0.1i \). We set \( B_{\text{init}} = B_6 \) where it is equally likely that the pedestrian is in mode \( M_1^{env} \) or mode \( M_2^{env} \). The belief update function \( \tau \) is defined such that the longer the pedestrian stay on the left side of the road, the probability that s/he is in mode \( M_1^{env} \) increases. Once the pedestrian starts crossing the road, we change the belief to \( B_0 \) where it is certain that the pedestrian is in mode \( M_2^{env} \). Specifically, for all \( i \), we let

\[
\tau(B_i, s, s') = \begin{cases} 
B_0 & \text{if } s' \in \{c_2, c_4, c_6, c_7\} \\
B_i & \text{if } i = 0 \text{ or } i = 8 \\
B_{i+1} & \text{otherwise}
\end{cases}
\]  \hspace{1cm} (19)

Both the expectation-based and the worst-case-base control policy synthesis as described in Section IV and V is implemented in MATLAB. The computation was performed on a MacBook Pro with a 2.8 GHz Intel Core 2 Duo processor.

First, we consider the expectation-based control policy synthesis (i.e., Problem 1). As outlined in Section IV we first construct the MDP that represents the complete system. After removing all the unreachable states, the resulting MDP contains 65 states and the product MDP contains 53 states. The computation time is summarized in Table I. Note that the computation, especially the product MDP construction, can be sped up significantly if a more efficient representation of DRA is used. The maximum expected probability for the system to satisfy \( \varphi \) is 0.9454. Examination of the resulting control policy shows that this maximum expected probability of satisfying \( \varphi \) can be achieved by applying action \( \alpha_1 \), i.e., decelerating, until the pedestrian crosses the street or the vehicle is not behind the pedestrian in the longitudinal direction, i.e., when the vehicle is in cell \( c_2 \) and the pedestrian is in cell \( c_j \) where \( j < i \). (Based on the expectation, the
probability that the pedestrian eventually crosses the road is 1 according to the probability measure defined in (1). If we include the belief B where B(M1env) = 1 and B(M2env) = 0, then the expectation-based optimal control policy is such that the vehicle applies α1 until either the pedestrian crosses the road, the vehicle is not behind the pedestrian in the longitudinal direction or the belief is updated to B, at which point, it applies α2. Once the vehicle reaches the destination c8, it applies α1 forever.

Next, we consider the worst-case-based control policy synthesis (i.e., Problem 2). In this case, the resulting AMDP contains 49 states and the product AMDP contains 53 states after removing all the unreachable states. The maximum worst-case probability for the system to satisfy ϕ is 0.9033. The resulting control policy is slightly more aggressive than the expectation-based policy. In addition to the cases where the expectation-based controller applies α2, the worst-case-based controller also applies α2 when the vehicle is in c0 and the pedestrian is in c3 and when the vehicle is in c2 and the pedestrian is in c6.

<table>
<thead>
<tr>
<th>MDP / AMDP</th>
<th>product MDP / AMDP</th>
<th>Control policy</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expectation</td>
<td>0.05</td>
<td>2.31</td>
<td>0.73</td>
</tr>
<tr>
<td>Worst-case</td>
<td>0.20</td>
<td>2.73</td>
<td>0.46</td>
</tr>
</tbody>
</table>

**TABLE 1**
TIME REQUIRED (IN SECONDS) FOR EACH STEP OF COMPUTATION.

Simulation results are shown in Figure 4 and Figure 5. The smaller (red) rectangle represents the pedestrian whereas the bigger (blue) rectangle represents the vehicle. The filled and unfilled rectangles represent their current positions and the trace of their trajectories, respectively. Notice that the vehicle successfully reaches its goal without colliding with the pedestrian, as required by its specification, with the worst-case-based controller being slightly more aggressive.

Finally, we would like to note that due to the structure of this example, the worst-case-based synthesis problem can be solved without having to deal with the belief space at all. As the environment cannot be in state c2, c4, c6, or c7 when it is in mode M1env and these states only have transitions among themselves, once the environment transitions to one of these states, we know for sure that it can only be in mode M1env and cannot change its mode anymore. In states c1, c3, and c5, the environment can be in either mode. Based on this structure and Remark 1, we can construct an AMDP that represents the complete system with smaller number of states than MAM constructed using the method described in Section V-B. Exploiting the structure of the problem to reduce the size of AMDP is subject to future work.

**VII. CONCLUSIONS AND FUTURE WORK**

We took an initial step towards solving POMDPs that are subject to temporal logic specifications. In particular, we considered the problem where the system interacts with its dynamic environment. A collection of possible environment
models are available to the system. Different models correspond to different modes of the environment. However, the system does not know in which mode the environment is. In addition, the environment may change its mode during an execution. Control policy synthesis was considered with respect to two different objectives: maximizing the expected probability and maximizing the worst-case probability that the system satisfies a given temporal logic specification.

Future work includes investigating methodologies to approximate the belief space with a finite set of representative points. This problem has been considered extensively in the POMDP literature. Since the value iteration used to obtain a solution to our expectation-based synthesis problem is similar to the value iteration used to solve POMDP problems where the expected reward is to be maximized, we believe that existing sampling techniques used to solve POMDP problems can be adapted to solve our problem. Another direction of research is to integrate methodologies for discrete state estimation to reduce the size of AMDP for the worst-case-based synthesis problem.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Tirthankar Bandyopadhyay for inspiring discussions.

REFERENCES


