Temporal Logic Motion Planning and Control With Probabilistic Satisfaction Guarantees

Morteza Lahijanian, *Student Member, IEEE*, Sean B. Andersson, *Member, IEEE*, and Calin Belta, *Member, IEEE*

***Abstract*—We describe a computational framework for automatic deployment of a robot with sensor and actuator noise from a temporal logic specification over a set of properties that are satisfied by the regions of a partitioned environment. We model the motion of the robot in the environment as a Markov decision process (MDP) and translate the motion specification to a formula of probabilistic computation tree logic (PCTL). As a result, the robot control problem is mapped to that of generating an MDP control policy from a PCTL formula. We present algorithms for the synthesis of such policies for different classes of PCTL formulas. We illustrate our method with simulation and experimental results.**

***Index Terms*—Motion planning, probabilistic computation tree logic (PCTL), stochastic control.**

I. Introduction

MOTION planning is a rich field, and many textbooks (e.g., [3]) describe the variety of available algorithms. Early studies of motion planning considered only the geometry of the robot and environment, together with simple point-to-point tasks. As the field has evolved, methods, such as tree-based techniques [4], [5] and layered planning [6], have made motion planning possible even for systems with complex dynamics [7]–[9]. However, the "classical" motion planning specifications remain simple and are given as "go from A to B and avoid obstacles," where A and B are two regions of interest in some environment.

Recently, there has been increasing interest in developing computational frameworks allowing planning for more complicated tasks. Increased expressivity is necessary in specifications that require reaching one of a set of goals ("go to either A or B"), convergence to a region ("reach A eventually and stay there for all future times"), visiting targets sequentially ("reach A, and then B, and then C"), surveillance ("reach A and then B infinitely often"), or the satisfaction of more complicated temporal and logic conditions about the reachability of regions of interest ("Never go to A. Do not go to B unless C or D were visited"). To accommodate such specifications, in [10]–[14], the authors suggested the use of temporal logics, such as linear temporal logic (LTL) and computation tree logic (CTL) as motion-specification languages. In [11], the authors augment the tree-search framework with LTL trajectory properties. In [12] and [13], the authors generate trajectories that satisfy LTL specifications on the sequence of regions that are visited by a robot by using controllers that could drive the robot between adjacent regions. In [14], the authors combine receding horizon planning with LTL motion planning. Generally, temporal logic motion planning and control are achieved based on algorithms that are inspired from model checking [15] and temporal logic games [16].

Most of the existing formal synthesis approaches are based on abstracting the robot motion in a partitioned environment to a finite transition system [15]. They operate under two fundamental assumptions. First, the transition system is either purely deterministic, in which each control action gives rise to a unique transition from each region, or purely nondeterministic, in which control actions enable possibly several transitions, with no information on the likelihoods of those transitions [17]. Second, the robot is able to determine its position in the environment precisely. Unfortunately, in real-world systems, noise in the actuators and sensors can invalidate both of these assumptions. In general, one must deal with a partially observed Markov decision process (POMDP) describing the motion of the robot in the environment. For basic region-to-region tasks, numerous algorithms exist to determine a robot control strategy for a POMDP model [18]–[20]. For more complicated tasks, probabilistic counterparts to temporal logics exist for specifying the task [21]–[23]. However, the problem of generating a control strategy for a POMDP by a formula in such a logic remains unsolved.

We focus on a simpler version of this problem. We consider a robot that moves in a partitioned environment by applying a given set of motion primitives allowing it to steer between adjacent regions. Because of sensor and actuation noise, while applying an available motion primitive at a region, the robot can transit to more than one adjacent region. We assume that the probabilities of these transitions are known and the robot can determine its current region precisely. In indoor environments, the latter assumption is not overly restrictive since simple environment modifications can be made to enforce it, such as placing a large number of radio-frequency identification (RFID) tags. In previous work, we solved this problem by abstracting the

Manuscript received November 12, 2010; revised May 24, 2011; accepted October 5, 2011. Date of publication November 21, 2011; date of current version April 9, 2012. This paper was recommended for publication by Associate Editor T. Simeon and Editor D. Fox upon evaluation of the reviewers' comments. This work was supported in part by Boston University through the National Science Foundation under Grant CNS-0834260 and Grant CMMI-0928776, the Army Research Office (ARO) under Grant W911NF-09-1-0088, the Air Force Office of Scientific Research (AFOSR) under Grant FA9550-09-1-0209, and the Office of Naval Research (ONR) under Grant ONR MURI N00014-09-1051. This paper was presented in part at the IEEE International Conference on Robotics and Automation, Anchorage, AK, 2010, and at the American Control Conference, San Francisco, CA, 2011

The authors are with the Department of Mechanical Engineering, Boston University, Boston, MA 02215 USA (e-mail: morteza@bu.edu; sanderss@bu.edu; cbelta@bu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2011.2172150

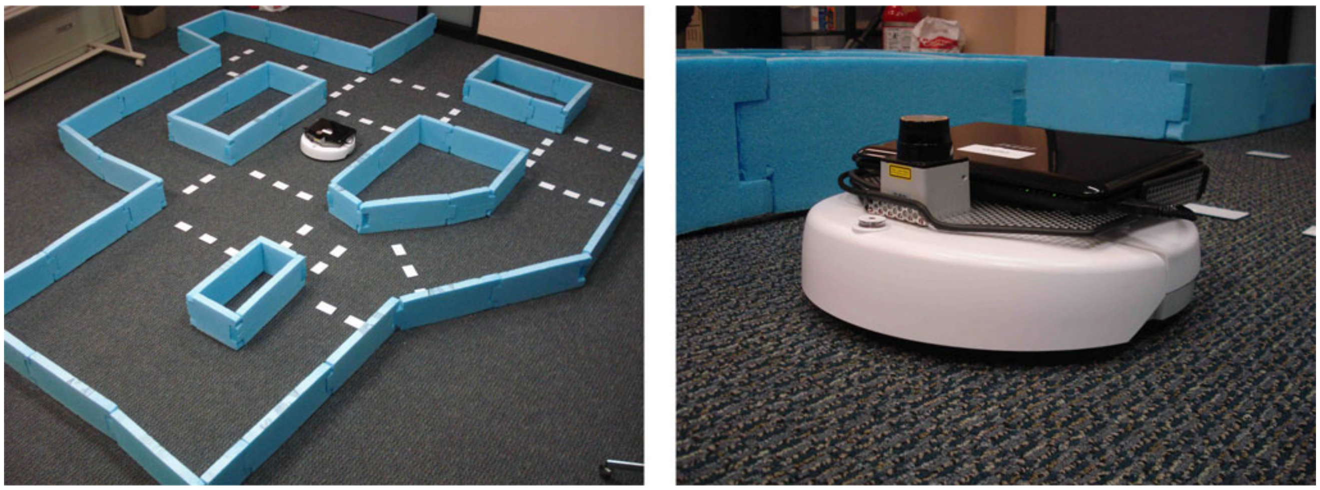

Fig. 1. Robotic indoor environment. (Left) iRobot Create mobile platform that is equipped with a laptop, a laser range finder, and RFID reader moves autonomously through the corridors and intersection of an indoor-like environment, whose topology can be easily reconfigured by moving the foam walls and (right) robot closeup.

robot motion to an MDP and deriving an algorithm to synthesize control strategies for a small segment of probabilistic computation tree logic (PCTL) formulas, namely those containing only a single temporal operator "until." To define the states of the MDP, we used the regions that are given by the partitioned environment and found the transition probabilities by using a Monte Carlo method. We derived the control synthesis algorithm through infinite horizon dynamic programming methods. A similar approach was taken in [24] to solve for a probabilistic reachability motion planning problem (equivalent to the basic "until" temporal operator available in the PCTL) with a different MDP abstraction technique. In particular, in [24], the authors combined a roadmap representation of the configuration space with a stochastic model of the robot motion to build a stochastic motion roadmap. While the underlying synthesis methods in that work and in this paper are similar, the framework that we defined in [1] and build on here allows for significantly more expressivity in the specification of the tasks. We extended our approach in [2] to support additional PCTL formulas with multiple and different temporal operators.

In this paper, we complete the effort that was started in [1] and [2] and present the entire MDP control synthesis algorithm from PCTL formulas that include Boolean operators, temporal operators, expected cost operators, and any combinations thereof. We can also accommodate nested probabilities, which allow for complex specifications, such as "Eventually reach A and then B with probability greater than 0.9, while always avoiding the regions from which the probabilities of converging to C is greater than 0.2" and "Eventually reach a region from which the probability of converging to A is 0.8, while minimizing the total amount of travel time." In short, given a specification as a PCTL formula, the algorithm returns the maximum probability or the minimum cost of satisfaction and a control strategy that achieves this probability or cost. Our algorithm uses subalgorithms corresponding to each temporal operator as the building blocks for the construction of a control strategy from a formula with multiple temporal operators. The most computationally expensive subalgorithm requires solving a linear programming problem.

The contribution of this study is threefold. First, we present the algorithms for MDP control synthesis from PCTL formulas in a unified way. While the building blocks of our control synthesis algorithm are based on an adaptation of the existing PCTL model checking algorithms [25], the synthesis approach to PCTL formulas with more than one operator and the framework are, to the best of our knowledge, novel and quite general. Second, we develop a framework for automatic robot deployment from temporal logic specifications with probabilistic satisfaction guarantees, which combines experiments, simulations, and statistical hypothesis testing with the synthesis algorithms that are mentioned earlier. Third, we illustrate the method in our Robotic InDoor Environment (RIDE) [26], in which an iRobot Create platform that is equipped with RFID readers, and a laser range finder moves autonomously through the corridors and intersections of an easily reconfigurable environment.

The remainder of the paper is organized as follows. In Section II, we formulate the problem and state our approach. In Section III, we formally define an MDP, a probability measure over paths of an MDP, and the specification language PCTL. The MDP control synthesis from PCTL formulas and the issues of conservatism and complexity of the algorithms are discussed in Section IV. The results of experimental and simulation case studies are included in Section V. The paper is concluded with final remarks in Section VI.

II. Problem Formulation and Approach

Consider a robot moving in a partitioned environment. We assume that the robot is programmed with a small set of feedback control primitives allowing it to move inside each region and from a region to an adjacent region. We make the natural assumption that these control primitives are not completely reliable. In other words, if, at a given region, a control primitive that is designed to take the robot to a specific adjacent region is used, it is possible that the robot will instead transition to a

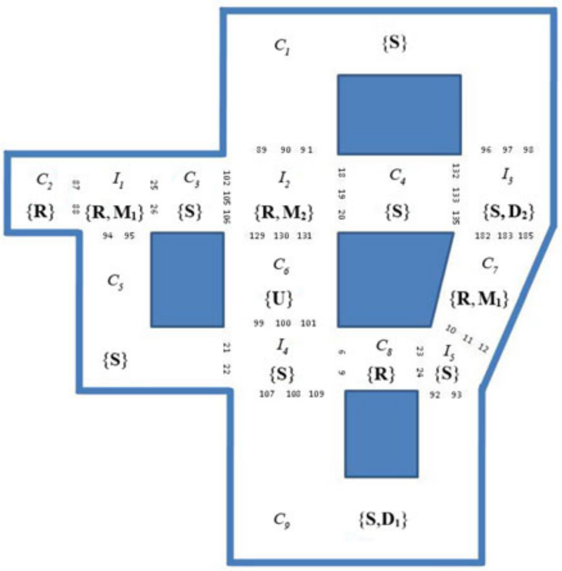

Fig. 2. Schematic representation of the environment from Fig. 1. Each region has a unique identifier ($C_1, \ldots, C_9$ for corridors and $I_1, \ldots, I_5$ for intersections, respectively). The properties that are satisfied at the regions are shown between curly brackets inside the regions $\mathbf{S}$ = *Safe*, $\mathbf{R}$ = *Relatively safe*, $\mathbf{U}$ = *Unsafe*, $\mathbf{M_1}$ = *Medical supply 1*, $\mathbf{M_2}$ = *Medical supply 2*, $\mathbf{D_1}$ = *Destination 1*, and $\mathbf{D_1}$ = *Destination 1*. The numbers that are shown at the boundaries between corridors and intersections indicate the positions and tag numbers of the RFID tags.

different adjacent region. We assume that the probabilities of these transitions are known and that the robot can precisely determine its current region. We consider the following problem.

Problem 1: Given a motion specification as a temporal logic statement about properties that are satisfied by the regions in a partitioned environment, find a robot control strategy that maximizes the probability of satisfying the specification.

Consider, e.g., an environment with the schematic representation that is shown in Fig. 2. The regions are corridors and intersections, which are identified by $C_1, \ldots, C_9$ and $I_1, \ldots, I_5$, respectively. There are six properties of interest about the regions: *Safe* (the robot can safely drive through a corridor or intersection with this property), *Relatively safe* (the robot can pass through the region but should avoid it if possible), *Unsafe* (the corresponding region should be avoided), *Medical supplies 1* and *2* (there are medical supplies of type 1 and 2 in the regions that are associated with these properties, respectively), and *Destinations 1* and *2* (regions to be visited). Some examples of temporal logic motion specifications are as follows.

Specification 1: Reach *Destination 1* by driving through either only *Safe* regions or through *Relatively safe* regions only if *Medical Supply 1* is available at such regions.

Specification 2: Reach *Destination 1* by going through the regions from which the probability of converging to a *Relatively safe* region is less than 0.50 and always avoiding *Unsafe* regions.

Specification 3: Reach *Destination 1* by driving through regions that are *Safe* or at which *Medical supply 2* is available, and then with the probability greater than or equal to 0.50 reach *Destination 2* by avoiding *Unsafe* regions and regions that are *Relatively safe* but do not have *Medical supply 1*.

Specification 4: Eventually, visit *Destination 1* and then, with the probability greater than 0.9, reach *Destination 2*, while avoiding *Unsafe* regions by minimizing the total amount of travel time.

It should be noted that Specification 1 requires the robot to visit only one destination, while Specification 2 involves the properties of regions and a probability, and Specification 3 requires to visit multiple destinations. Specification 4 combines cost and probability for multiple destinations.

As will be made clear later, the temporal logic specification will be a formula of PCTL. A robot control strategy will be defined as an assignment of a control primitive to each region of the environment that the robot visits given the history of the visited regions. Since the outcome of a control primitive is characterized probabilistically, the satisfaction of the specification is defined in a probabilistic sense. Among all the possible control strategies, our solution to the aforementioned problem will have the highest rate of success.

Central to our approach is a representation of the motion of the robot in the environment as an MDP (see Section III-A). For the construction of such a model over a given environment and robot motion and sensor models, we use a combination of physical experiments, simulations, and statistical hypothesis testing to determine the success/failure rate of each action (feedback control primitive). To ensure that the transitions are Markovian, we augment the state space by choosing an ordered set of partition regions for each state in the MDP. As discussed earlier, we assume that, despite the stochastic nature of its motion, the robot can determine its current region precisely. This can be achieved in a variety of ways. For example, in the RIDE, we place a set of RFID tags that uniquely identify the regions. Given this model, Problem 1 reduces to generating a control strategy for an MDP, such that the probability of satisfying a PCTL formula is maximized. This problem is treated in Section IV.

It should be noted that the problem that we consider in this paper mainly focuses on high-level mission planning given low-level controllers. Even though the design of the motion primitives (low-level control primitives), which are required to enable transitions between regions and in the construction of an MDP model of the motion of the robot in the environment, is an important and challenging problem itself, it is out of the scope of this paper. However, we argue that the proposed framework is quite general and can be used to control robots with complex dynamics provided some motion primitives. The performance of the algorithm is independent of the preciseness of the design of the low-level control primitives. Nevertheless, the maximum probability value of satisfaction of the specification is directly related to the reliability of such controls. Generally, as the motion primitives become more reliable, the maximum probability of satisfaction increases.

III. Preliminaries

A. Markov Decision Process

Given a set Q, let $|Q|$ and 2^Q denote its cardinality and power set, respectively.

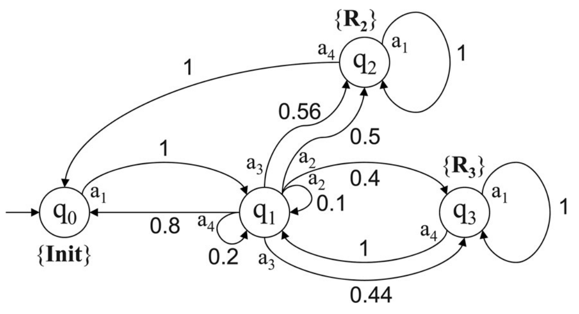

Fig. 3. Four-state MDP.

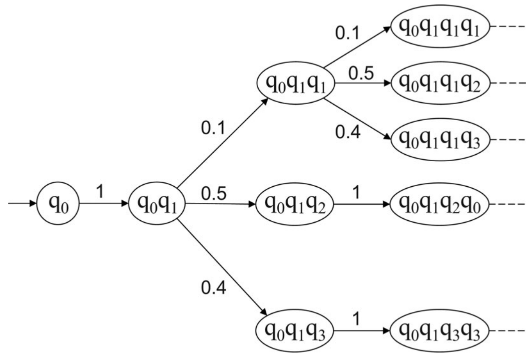

Fig. 4. Fragment of DTMCs $\mathcal{D}_{\mu_1}$ for control policy μ_1.

Definition 1 (Markov Decision Process): An MDP is a tuple, i.e., $\mathcal{M} = (Q, q_0, \text{Act}, \text{Steps}, \Pi, L, c)$, where we have the following.

1) Q is a finite set of states.
2) $q_0 \in Q$ is the initial state.
3) Act is a set of actions.
4) Steps: $Q \to 2^{\text{Act}\times\Sigma(Q)}$ is a transition probability function, where $\Sigma(Q)$ is the set of all discrete probability distributions over the set Q.
5) Π is a finite set of atomic propositions.
6) $L : Q \to 2^{\Pi}$ is a labeling function assigning to each state possibly several elements of Π.
7) Cost : $Q \times \text{Act} \to \mathbb{R}^{\geq 0}$ is a cost function.

The set of actions that are available at $q \in Q$ is denoted by $\mathcal{A}(q)$. The function Steps is often represented as a matrix with $|Q|$ columns and $\sum_{i=0}^{|Q|-1} |\mathcal{A}(q_i)|$ rows. For each action $a \in \mathcal{A}(q_i)$, we denote the probability of transitioning from the states q_i to q_j under the action a as $\sigma_a^{q_i}(q_j)$ and the corresponding probability distribution function as $\sigma_a^{q_i}$. Each $\sigma_a^{q_i}$ corresponds to one row in the matrix representation of Steps. The cost function "cost" assigns to each state $q \in Q$ and action $a \in A(q)$ a nonnegative cost cost(q, a).

To illustrate these definitions, a simple MDP is shown in Fig. 3. The actions that are available at each state are $\mathcal{A}(q_0) = \{a_1\}$, $\mathcal{A}(q_1) = \{a_2, a_3, a_4\}$, and $\mathcal{A}(q_2) = \mathcal{A}(q_3) = \{a_1, a_4\}$. The labels are $L(q_0) = \{\mathbf{Init}\}$, $L(q_2) = \{\mathbf{R_2}\}$, and $L(q_3) = \{\mathbf{R_3}\}$. The matrix representation of "Steps" is given by

$$\text{Steps} = \begin{array}{c} \underline{q_0; a_1} \\ q_1; a_2 \\ q_1; a_3 \\ \underline{q_1; a_4} \\ q_2; a_1 \\ \underline{q_2; a_4} \\ q_3; a_1 \\ q_3; a_4 \end{array} \left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \hline 0 & 0.1 & 0.5 & 0.4 \\ 0 & 0 & 0.56 & 0.44 \\ 0.8 & 0.2 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array} \right).$$

In this example, the cost of all the state–action pairs is zero.

B. Paths, Control Policies, and Probabilistic Measures

A path ω through an MDP is a sequence of states, i.e., $\omega = q_0 \xrightarrow{(a_0, \sigma_{a_0}^{q_0}(q_1))} q_1 \xrightarrow{(a_1, \sigma_{a_1}^{q_1}(q_2))} \cdots q_i \xrightarrow{(a_i, \sigma_{a_i}^{q_i}(q_{i+1}))} q_{i+1} \ldots$, where each transition is induced by a choice of action at the current step $i \geq 0$. We denote the ith state of a path ω by $\omega(i)$ and the set of all finite and infinite paths by Path$^{\text{fin}}$ and Path, respectively.

A control policy defines a choice of actions at each state of an MDP. Control policies are also known as schedulers or adversaries and are formally defined as follows.

Definition 2 (Control Policy): A control policy μ of an MDP model $\mathcal{M}$ is a function mapping a finite path, i.e., $\omega^{\text{fin}} = q_0 q_1 q_2 \cdots q_n$, of $\mathcal{M}$ onto an action in $\mathcal{A}(q_n)$. In other words, a policy is a function, i.e., $\mu : \text{Path}^{\text{fin}} \to \text{Act}$, that specifies for every finite path, the next action to be applied. If a control policy depends only on the last state of ω^{fin}, it is called a stationary policy.

For each policy μ, a probability measure Prob$_\mu$ over the set of all paths (under μ) Path$_\mu$ is induced. It is constructed through an infinite-state Markov chain as follows. Under a policy μ, an MDP becomes a Markov chain that is denoted $\mathcal{D}_\mu$ whose states are the finite paths of the MDP $\mathcal{M}$. There is a one-to-one correspondence between the paths of $\mathcal{D}_\mu$ and the set of paths Path$_\mu$ in the MDP. Hence, a probability measure Prob$_\mu$ over the set of paths Path$_\mu$ can be defined by setting the probability of $\omega^{\text{fin}} \in \text{Path}_\mu^{\text{fin}}$ equal to the product of the corresponding transition probabilities in $\mathcal{D}_\mu$.

Next, we define the cylinder set $C(\omega^{\text{fin}})$ as

$$C(\omega^{\text{fin}}) = \{\omega \in \text{Path}_\mu | \omega^{\text{fin}} \text{ is a prefix of } \omega\}$$

i.e., the set of all infinite paths with prefix ω^{fin}. The probability measure on the smallest σ-algebra over Path$_\mu$ containing $C(\omega^{\text{fin}})$ for all $\omega^{\text{fin}} \in \text{Path}_\mu^{\text{fin}}$ is the unique measure satisfying

$$\text{Prob}_\mu(C(\omega^{\text{fin}})) = \text{Prob}_\mu(\omega^{\text{fin}}) \;\; \forall \omega^{\text{fin}} \in \text{Path}_\mu^{\text{fin}}. \qquad (1)$$

To illustrate this measure, consider the MDP that is shown in Fig. 3 and the stationary control policy that is defined by the mapping

$$\mu_1(\cdots q_0) = a_1, \;\; \mu_1(\cdots q_1) = a_2$$
$$\mu_1(\cdots q_2) = a_4, \;\; \mu_1(\cdots q_3) = a_1$$

where "$\cdots q_i$" denotes any finite path terminating in q_i. The initial fragment of the resulting Markov chain is shown in Fig. 4. From this fragment, it is easy to see that the probability of the finite path $q_0 q_1 q_2$ is $\text{Prob}_{\mu_1}(q_0 q_1 q_2) = 0.5$. Under μ_1, the set of all infinite paths with this prefix is

$$C(q_0q_1q_2) = \{\overline{q_0q_1q_2},\ q_0q_1q_2q_0\overline{q_1},\ q_0q_1q_2q_0q_1\overline{q_3}, \ldots\}$$

where the sequence under the overline is repeated infinitely. According to (1), we have that

$$\text{Prob}_{\mu_1}(C(q_0q_1q_2)) = \text{Prob}_{\mu_1}(q_0q_1q_2) = 0.5.$$

C. Probabilistic Computation Tree Logic

We use PCTL [27], a probabilistic extension of CTL that includes a probabilistic operator $\mathcal{P}$, to write specifications of the MDP.

Definition 3 (Syntax of PCTL): PCTL formulas are the state formulas, which can be recursively defined as follows:

$$\phi ::= \text{true} \mid \pi \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie\, p}[\psi] \mid \mathcal{E}_{\bowtie\, c}[\phi] \text{ state formulas}$$
$$\psi ::= X\phi \mid \phi\, \mathcal{U}^{\leq k}\, \phi \mid \phi\, \mathcal{U}\, \phi \text{ path formulas}$$

where $\pi \in \Pi$ is an atomic proposition; $\neg$ (negation) and $\wedge$ (conjunction) are the Boolean operators; $\bowtie \in \{\leq, <, \geq, >\}$; $p \in [0, 1]$, $c \in [0, \infty]$; and $k \in \mathbb{N}$.

State formulas ϕ are evaluated over the states of an MDP, while the path formulas ψ are assessed over paths and only allowed as the parameter of the $\mathcal{P}_{\bowtie\, p}[\psi]$ operator. Intuitively, a state q satisfies $\mathcal{P}_{\bowtie\, p}[\psi]$ if there exists a control policy that will ensure that the probability of all paths from q under that policy satisfying ψ is in the range $\bowtie p$. Similarly, if q satisfies $\mathcal{E}_{\bowtie c}[\phi]$, then the expected cost of reaching a state satisfying ϕ is $\bowtie c$. Temporal logic operators X ("next," also denoted by $\bigcirc$), $\mathcal{U}^{\leq k}$ ("bounded until"), and $\mathcal{U}$ ("until") are allowed in the path formulas.

Definition 4 (Semantics of PCTL): For any state $q \in Q$, the satisfaction relation $\models$ is defined inductively as follows:

1) $q \models \text{true}$ for all $q \in Q$;
2) $q \models \pi \Leftrightarrow \pi \in L(q)$;
3) $q \models (\phi_1 \wedge \phi_2) \Leftrightarrow q \models \phi_1 \wedge q \models \phi_2$;
4) $q \models \neg\phi \Leftrightarrow q \not\models \phi$;
5) $q \models \mathcal{P}_{\bowtie p}[\psi] \Leftrightarrow p^q_\mu \bowtie p$;
6) $q \models \mathcal{E}_{\bowtie c}[\phi] \Leftrightarrow e^q_\mu \bowtie c$.

Here, p^q_μ is the probability of all the infinite paths that start from q and satisfy ψ under policy μ, and $e^q_\mu(\phi)$ denotes the total expected cost of reaching a state that satisfies ϕ from q under policy μ. Moreover, for any path $\omega \in$ Path

1) $\omega \models X\phi \Leftrightarrow \omega(1) \models \phi$;
2) $\omega \models \phi_1\, \mathcal{U}^{\leq k}\, \phi_2 \Leftrightarrow \exists\, i \leq k,\ \omega(i) \models \phi_2 \wedge \omega(j) \models \phi_1 \forall j < i$;
3) $\omega \models \phi_1\, \mathcal{U}\, \phi_2 \Leftrightarrow \exists\, k \geq 0,\ \omega \models \phi_1\, \mathcal{U}^{\leq k}\, \phi_2$.

The other standard logical operators false, $\vee$ (disjunction), and $\rightarrow$ (implication) can be easily derived from the basic syntax given earlier:

$$\text{false} \equiv \neg\text{true}$$
$$\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$$
$$\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2.$$

We can also define the path formula operators $\Diamond$, "eventually," (also denoted by F), and $\Diamond^{\leq k}$, "bounded eventually," ($F^{\leq k}$) as

$$\mathcal{P}_{\bowtie p}[\Diamond\phi] \equiv \mathcal{P}_{\bowtie p}[\text{true}\,\mathcal{U}\phi]$$
$$\mathcal{P}_{\bowtie p}[\Diamond^{\leq k}\phi] \equiv \mathcal{P}_{\bowtie p}[\text{true}\,\mathcal{U}^{\leq k}\phi].$$

Intuitively, $\Diamond\phi$ means that ϕ is eventually satisfied. Similarly, $\Diamond^{\leq k}\phi$ means that ϕ is satisfied within k time units.

Other common temporal operators are $\Box$, "always" (also denoted by G), and $\Box^{\leq k}$, "bounded always" ($G^{\leq k}$). Defining these operators in PCTL is slightly more complex than in nonprobabilistic temporal logics. It can be done using the fact that, for a state q, policy μ and path formula ψ, $p^q_\mu(\neg\psi) = 1 - p^q_\mu(\psi)$. Then, it can be shown that [27]

$$\mathcal{P}_{\leq p}[\Box\phi] \Leftrightarrow \mathcal{P}_{>1-p}[\Diamond\neg\phi].$$

More generally

$$\mathcal{P}_{\bowtie p}[\Box\phi] \equiv \mathcal{P}_{\bar{\bowtie} p}[\Diamond\neg\phi]$$
$$\mathcal{P}_{\bowtie p}[\Box^{\leq k}\phi] \equiv \mathcal{P}_{\bar{\bowtie} p}[\Diamond^{\leq k}\neg\phi]$$

where $\bar{\leq} \equiv \geq$, $\bar{<} \equiv >$, $\bar{\geq} \equiv \leq$, and $\bar{>} \equiv <$.

IV. Probabilistic Computation Tree Logic Control Synthesis

In this section, we consider the following problem.

Problem 2: Given an MDP model $\mathcal{M}$ and a PCTL specification formula ϕ, find a control policy that maximizes the probability of satisfying ϕ.

Our control synthesis algorithm takes a PCTL formula ϕ and an MDP $\mathcal{M}$ and returns both the optimal probability of satisfying ϕ and the corresponding control policy. The basic algorithm proceeds by constructing the parse tree for ϕ and treating each operator in the formula separately. It should be noted that, in general, we are interested in finding the control policy that produces the maximum/minimum probability of satisfying the given specification. Such PCTL formulas have the form $\mathcal{P}_{\max=?}[\psi]$ and $\mathcal{P}_{\min=?}[\psi]$. For the PCTL formulas of the form $\mathcal{P}_{\bowtie p}[\psi]$, we still use the algorithms that return optimal policies (with the exception of the case that is discussed in Section IV-B). For these formulas, we first find the control policy μ and then check whether $p^q_\mu(\phi)$ satisfies the bound $\bowtie p$. For the case $\bowtie \in \{>, \geq\}$, we use the policy that maximizes the probability of satisfaction. Similarly, we determine the policy corresponding to the minimum probability when $\bowtie \in \{<, \leq\}$.

There is a set of unique algorithms for control synthesis of each path formula. Thus, we categorize the PCTL formulas into two groups: *simple* and *complex*. *Simple* formulas are those that include only one $\mathcal{P}$- or $\mathcal{E}$-operator. An example of this class of specifications is Specification 1 in Section II, which directly translates to the PCTL formula

$$\phi_1 : \mathcal{P}_{\max=?}\,[(\mathbf{S} \vee (\mathbf{R} \wedge \mathbf{M_1}))\,\mathcal{U}\,\mathbf{D_1}\,]. \tag{2}$$

The control algorithm for such formulas is discussed in Sections IV-A. *Complex* PCTL formulas are the ones that consist of more than one path formulas. Specifications 2–4 in Section II, which naturally translate to the PCTL formulas

$$\phi_2 : \mathcal{P}_{\max=?}\left[(\mathcal{P}_{<0.50}[X\,\mathbf{R}] \wedge \neg\mathbf{U})\,\mathcal{U}\,\mathbf{D_1}\right] \quad (3)$$

$$\phi_3 : \mathcal{P}_{\max=?}\left[(\mathbf{S} \vee \mathbf{M_2})\,\mathcal{U}\,(\mathbf{D_1} \wedge \mathcal{P}_{\geq 0.50}[\neg\mathbf{U} \wedge \neg(\mathbf{R} \wedge \neg\mathbf{M_1})\,\mathcal{U}\,\mathbf{D_2}])\right] \quad (4)$$

$$\phi_4 : \mathcal{E}_{\min=?}\left[\mathbf{D_1} \wedge \mathcal{P}_{>0.9}[\neg\mathbf{U}\,\mathcal{U}\,\mathbf{D_2}])\right] \quad (5)$$

belong to this class of specifications. To find a control strategy for these formulas, we use the algorithms for simple formulas as building blocks. Our approach to complex specifications is discussed in Section IV-B.

A. Simple Probabilistic Computation Tree Logic Formulas

The optimal control synthesis algorithms for these formulas are derived by making a few modifications to an existing model checking algorithm [27]. We discuss these algorithms later and make the connection between these algorithms and the maximum reachability probability problem [28] and the stochastic shortest path problem [29].

1) Next Operator: For the "next" temporal operator, we present two algorithms. One finds the optimal control strategy, and the other determines all the satisfying policies. For the PCTL formulas that include only one $\mathcal{P}$-operator, the optimal control strategy algorithm is always used. For the formulas that include more than one probabilistic operator, both algorithms are used. This becomes clear in Section IV-B.

a) Next Optimal—$\phi = \mathcal{P}_{\max=?}[X\phi_1]$: For this operator, we need to determine the action that produces the maximum probability of satisfying $X\phi_1$ at each state. Thus, we only need to consider the immediate transitions at each state, and the problem reduces to the following:

$$x^*_{q_i} = \max_{a \in \mathcal{A}(q_i)} \sum_{q_j \in \mathrm{Sat}(\phi_1)} \sigma_a^{q_i}(q_j)$$

$$\mu^*(q_i) = \arg\max_{a \in \mathcal{A}(q_i)} \sum_{q_j \in \mathrm{Sat}(\phi_1)} \sigma_a^{q_i}(q_j)$$

where $x^*_{q_i}$ denotes the optimal probability of satisfying ϕ at the state $q_i \in Q$, $\mathrm{Sat}(\phi_1)$ is the set of states that satisfy ϕ_1, and μ^* represents the optimal policy.

To solve the aforementioned maximization problem, we define a state-indexed vector $\overline{\phi_1}$ with entries $\overline{\phi_1}(q_i)$ equal to 1 if $q_i \vDash \phi_1$ and 0 otherwise. To compute the maximum probability, first, the matrix "Steps" is multiplied by $\overline{\phi_1}$. The result is a vector whose entries are the probabilities of satisfying $X\phi_1$, where each row corresponds to a state–action pair. Then, the maximization operation is performed on this vector, which selects the maximum probabilities and the corresponding actions at each state. The resulting control strategy is stationary, and the complexity of achieving it is one matrix–vector multiplication followed by a 1-D search.

Similarly, for the case of $\phi = \mathcal{P}_{\min=?}[X\phi_1]$, we need to determine the actions that produce the minimum probability of satisfying $X\phi_1$. This problem is solved with the same method that is described earlier, but instead of the maximization step, a minimization operation is performed.

To demonstrate this method, consider the MDP in Fig. 3 and the formula, i.e., $\phi = \mathcal{P}_{\max=?}[X(\neg\mathbf{R_3})]$. The property $(\neg\mathbf{R_3})$ is satisfied at states q_0, q_1, and q_2; thus, $\overline{\neg\mathbf{R_3}} = (1\;1\;1\;0)^T$. Then, Steps $\cdot\, \overline{\neg\mathbf{R_3}} =$

$$\begin{array}{c} \underline{q_0;a_1} \\ q_1;a_2 \\ q_1;a_3 \\ \underline{q_1;a_4} \\ q_2;a_1 \\ \underline{q_2;a_4} \\ q_3;a_1 \\ q_3;a_4 \end{array}
\left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ \hline 0 & 0.1 & 0.5 & 0.4 \\ 0 & 0 & 0.56 & 0.44 \\ 0.8 & 0.2 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array}\right)
\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} =
\left(\begin{array}{c} 1 \\ \hline 0.6 \\ 0.56 \\ 1 \\ \hline 1 \\ 1 \\ \hline 0 \\ 1 \end{array}\right).$$

Thus, $x_{q_i} = 1$ for $i = 0, \ldots, 3$ and the optimal stationary policy is $\mu^*(q_0) = a_1$, $\mu^*(q_1) = a_4$, $\mu^*(q_2) = a_1$ or a_4, and $\mu^*(q_3) = a_4$.

b) Next all—$\mathcal{P}_{\bowtie p}[X\phi_1]$: Here, we are interested in finding all the policies that satisfy the formula. The algorithm is the same as the one for Optimal Next (see Section IV-A1a) up to the maximization step. After obtaining the vector Steps $\cdot\,\overline{\phi_1}$, which includes the probabilities of satisfying $X\phi_1$ for each state–action pair, we eliminate the state–action pairs whose probabilities are not in the range of $\bowtie p$. This operation determines all the states, actions, and their corresponding probabilities that satisfy $\mathcal{P}_{\bowtie p}[X\phi_1]$. This algorithm is only used in complex PCTL formulas, as explained in Section IV-B.

To illustrate this algorithm, consider the example in Section IV-A1a with the formula, i.e., $\phi = \mathcal{P}_{\geq 0.6}[X(\neg R_3)]$. All satisfying actions at states q_0, q_1, q_2, and q_3 are $\{a_1\}$, $\{a_2, a_4\}$, $\{a_1, a_4\}$, and $\{a_4\}$, respectively.

2) Bounded Until Operator: For this operator, we also introduce two algorithms: optimal and stationary. The optimal algorithm produces a history-dependent control policy. The stationary algorithm results in a stationary policy and is used only for nested formulas.

a) Bounded Until Optimal—$\phi = \mathcal{P}_{\max=?}[\phi_1\mathcal{U}^{\leq k}\phi_2]$: To find the probabilities $p^q_{\max}(\phi_1\mathcal{U}^{\leq k}\phi_2)$, we first group the MDP states into three subsets: states that always satisfy the specification Q^{yes}, states that never satisfy the specification Q^{no}, and the remaining states $Q^?$:

$$Q^{\mathrm{yes}} = \mathrm{Sat}(\phi_2)$$
$$Q^{\mathrm{no}} = Q \setminus (\mathrm{Sat}(\phi_1) \cup \mathrm{Sat}(\phi_2))$$
$$Q^? = Q \setminus (Q^{\mathrm{yes}} \cup Q^{\mathrm{no}}).$$

Trivially, the probabilities of the states in Q^{yes} and in Q^{no} are 1 and 0, respectively. The probabilities for the remaining states $q_i \in Q^?$ are defined recursively. If $k = 0$, then $p^{q_i}_{\max}(\phi_1\mathcal{U}^{\leq k}\phi_2) = 0\ \forall q_i \in Q^?$. For $k > 0$

$$x^k_{q_i} = \max_{a \in \mathcal{A}(q_i)} \left(\sum_{q_j \in Q^?} \sigma_a^{q_i}(q_j) x^{k-1}_{q_i} + \sum_{q_j \in Q^{\mathrm{yes}}} \sigma_a^{q_i}(q_j) \right) \quad \forall q_i \in Q^?$$

$$\mu^{*^k}(q_i) = \arg \max_{a\in\mathcal{A}(q_i)} \left(\sum_{q_j\in Q^?} \sigma_a^{q_i}(q_j) x_{q_i}^{k-1} + \sum_{q_j\in Q^{\text{yes}}} \sigma_a^{q_i}(q_j) \right) \quad \forall q_i \in Q^?$$

where $x_{q_i}^k$ and $\mu^{*^k}(q_i)$ denote the probability of satisfying ϕ and the corresponding optimal action at the state $q_i \in Q^?$ at time step k, respectively.

Thus, the computation of $p_{\max}^q(\phi_1\mathcal{U}^{\le k}\phi_2)$ can be carried out in k iterations, each similar to the process that is described for Optimal Next (see Section IV-A1a). The additional step here is that after each maximization operation, the entries of the resultant vector corresponding to states Q^{yes} and Q^{no} are replaced with 1 and 0, respectively. This step is performed to guarantee that the state-indexed vector always carries the correct probabilities. The complexity of this algorithm is k matrix–vector multiplication and k maximization operations. The overall policy is time dependent. That is, for each time index k, an action is assigned to each satisfying state.

The bounded until the minimization problem, i.e., $\phi = \mathcal{P}_{\min=?}[\phi_1\mathcal{U}^{\le k}\phi_2]$, requires to find the probabilities $p_{\min}^q(\phi_1\mathcal{U}^{\le k}\phi_2)$. This problem is solved with a similar method presented earlier, where instead of maximization, a minimization operation is carried out at each iteration.

To illustrate the optimal algorithm for "bounded until," again consider the MDP in Fig. 3 and the PCTL formula, i.e., $\phi = \mathcal{P}_{\max=?}[\text{true}\,\mathcal{U}^{\le 2}\mathbf{R_3}]$. By inspection, we have $Q^{\text{yes}} = \{q_3\}$, $Q^{\text{no}} = \emptyset$, and $Q^? = \{q_0, q_1, q_2\}$. By following the method presented earlier, we compute $\overline{x}^1 = (0\ 0.44\ 0\ 1)^T$ and $\mu^{*^1}(q_1) = a_3$. This means that there exists only one state in $Q^?$ that satisfies the formula in one step or less, q_1 with the probability 0.44, and action a_3. Another iteration results in $\overline{x}^2 = (0.44\ 0.444\ 0\ 1)^T$ and $\mu^{*^2}(q_0) = a_1$ and $\mu^{*^2}(q_1) = a_2$. Thus, q_1 satisfies the formula with the maximum probability of 0.444 in two steps or less with selection of actions a_2 and a_3 in the first and second time steps, respectively. Moreover, q_0 satisfies the formula with the probability 0.44 in two steps with the selection of actions a_1 at q_0 in the first time step and a_3 at q_1 in the second time step.

b) Bounded Until Stationary—$\phi = \mathcal{P}_{\bowtie p}[\phi_1\mathcal{U}^{\le k}\phi_2]$: Here, we introduce a suboptimal algorithm for the $\mathcal{U}^{\le k}$ operator, which produces a stationary control policy. This algorithm is used for control synthesis of nested formulas, where a stationary policy is required.

The algorithm is the same as the one for "Bounded Until Optimal" (see Section IV-A2a) with the exception that the optimal actions determined at each iteration are fixed for the remaining iterations. For instance, consider the example in Section IV-A2a with the formula $\mathcal{P}_{>0.4}[\Diamond^{\le 2}\mathbf{R_3}]$. After the first iteration, we find that $\overline{x}^1 = (0\ 0.44\ 0\ 1)^T$ and $\mu(q_1) = a_3$. For the next iteration, we only use action a_3 at q_1, which results in $\overline{x}^2 = (0.44\ 0.44\ 0\ 1)^T$ with the policy, i.e., $\mu(q_0) = a_1$ and $\mu(q_1) = a_3$. Thus, the states q_0 and q_1 satisfy the formula with the stationary policy, i.e., $\mu(q_0) = a_1$ and $\mu(q_1) = a_3$.

For the PCTL formulas of the form, i.e., $\phi = \mathcal{P}_{\bowtie p}[\phi_1\mathcal{U}^{\le k}\phi_2]$, it is theoretically possible to find all the satisfying policies. This becomes important for completeness of the solution for nested formulas (see Section IV-B1). However, it only can be achieved by enumerating every satisfying path and leads to exponential growth in the complexity of the algorithm. Thus, for large MDPs and time index k, finding all the satisfying policies might be impracticable. For this reason, we only use the optimal and stationary algorithms that are given earlier for $\mathcal{U}^{\le k}$.

Algorithm 1 Find $Q^{no} = \{q \in Q |\, p_q^{max}(\phi_1\mathcal{U}\phi_2) = 0\}$

```
$R := Sat(\phi_2)$
$done :=$ false
while $done =$ false do
    $R' := R \cup \{q \in Sat(\phi_1) | \exists a \in \mathcal{A}(q), \exists q' \in R$ such that $\sigma_a^q(q') > 0\}$
    if $R' = R$ then
        $done :=$ true
    end if
    $R := R'$
end while
return $Q \setminus R$
```

3) Unbounded Until Operator—$\phi = \mathcal{P}_{\max=?}[\phi_1\mathcal{U}\phi_2]$: Here, we are interested in computing probabilities $p_{\max}^q(\phi_1\mathcal{U}\phi_2)$ over all policies and finding the control strategy that gives rise to these optimal probabilities. To solve this problem, again, begin by dividing Q into the three subsets: Q^{yes} (states satisfying the formula with the probability 1), Q^{no} (states satisfying the formula with the probability 0), and $Q^?$ (the remaining states). We find Q^{no} by using Algorithm 1 [27].

The computation of optimal probabilities for the states in $Q^?$ is in fact the maximal reachability probability problem [28]. Thus, we can compute these probabilities by solving the following linear programming problem over the set of variables $\{x_{q_i} | q_i \in Q^?\}$:

$$\text{minimize} \sum_{q_i\in Q^?} x_{q_i} \text{ subject to:}$$

$$x_{q_i} \ge \sum_{q_j\in Q^?} \sigma_a^{q_i}(q_j) \,.\, x_{q_j} + \sum_{q_j\in Q^{\text{yes}}} \sigma_a^{q_i}(q_j)$$

for all $q_i \in Q^?$ and $(a, \sigma_a) \in \text{Steps}(q_i)$.

The problem admits a unique optimal solution, and the actions that give rise to this optimal solution at every state can be identified. Hence, the stationary control policy that produces the maximum probability that satisfies the specification can be obtained. The aforementioned linear programming problem can be solved using classical techniques, such as the simplex method, ellipsoid method [30], or value iteration. The complexity is a polynomial of the size of the MDP.

We solve the case of the minimum probability (i.e., $\phi = \mathcal{P}_{\min=?}[\phi_1\mathcal{U}\phi_2]$) in an analogous fashion, using Algorithm 2 in lieu of Algorithm 1 to find the set Q^{no}. This change is necessary because when computing $p_{\min}^q$, Q^{no} contains all those states that satisfy the formula with the probability 0 for *some* policy μ, while in the computation of $p_{\max}^q$, Q^{no} is the set of states that never satisfy the formula under *any* policy.

Algorithm 2 Find $Q^{no} = \{q \in Q | p_q^{min}(\phi_1 \mathcal{U} \phi_2) = 0\}$

```
$R := Sat(\phi_2)$
$done :=$ **false**
**while** $done =$ **false do**
    $R' := R \cup \{q \in Sat(\phi_1) | \forall a \in \mathcal{A}(q), \exists q' \in R$ such that $\sigma_a^q(q') > 0\}$
    **if** $R' = R$ **then**
        $done :=$ **true**
    **end if**
    $R := R'$
**end while**
**return** $Q \setminus R$
```

To illustrate this method, again consider the MDP in Fig. 3 with the specification $\mathcal{P}_{\max=?}[\neg \mathbf{R}_3 \,\mathcal{U}\, \mathbf{R}_2]$. Since state q_2 is the only one that satisfies the formula with the probability 1 and q_3 is the only one that fails the formula with the probability 1, we have the $Q^{\text{yes}} = \{q_2\}$, $Q^{\text{no}} = \{q_3\}$, and $Q^? = \{q_0, q_1\}$. From this, we have that $x_{q_2} = 1$ and $x_{q_3} = 0$. The solution to the linear optimization problem can be found to be $x_{q_0} = x_{q_1} = 0.56$ under the policy, i.e., $\mu(q_0) = a_1$ and $\mu(q_1) = a_3$.

The "until" operator is in fact the same as $\mathcal{U}^{\leq k}$ as $k \to \infty$. With this approach, the algorithm that is given in Section IV-A2a can also be used to solve this problem. Convergence to a solution with this method is guaranteed by value iteration. Thus, it is theoretically possible to find all of the satisfying policies for the formulas of the form $\mathcal{P}_{\bowtie p}[\phi_1 \mathcal{U} \phi_2]$. However, as explained in Section IV-A2, because of high complexity, finding all the solutions is not tractable for even moderately sized problems. Thus, we only use the optimal algorithm that is given earlier to solve for such PCTL formulas.

4) Expected Cost Operator—$\mathcal{E}_{\min=?}[\phi]$: For the PCTL "expected cost" operator, we are required to compute the cost $e^q_{\min}(\phi)$ and find the corresponding control policy.

To proceed, first, we compute the sets of states Q^0 and Q^∞, which have a minimum expected cost of 0 and ∞, respectively, of reaching a state satisfying ϕ. Thus, Q^0 is the set of those states that satisfy ϕ, and Q^∞ contains all the states for which $p^q_\mu(\diamondsuit\phi) < 1$ for some policies. Hence, we can identify Q^∞ by using the model checking of the PCTL formula $\mathcal{P}_{<1}[\diamondsuit\phi]$.

The problem of computation of the minimum expected cost for the remaining states in $Q^? = Q \setminus (Q^0 \cup Q^\infty)$ is in fact the stochastic shortest path problem that is introduced in [29] and modified in [28]. Thus, these probabilities can be obtained by solving the following linear programming problem over the set of variables $\{y_{q_i} \mid q_i \in Q^?\}$:

$$\text{maximize} \sum_{q_i \in Q^?} y_{q_i} \text{ subject to :}$$

$$y_{q_i} \leq \text{cost}(q_i, a) + \sum_{q_j \in Q^?} \sigma_a^{q_i}(q_j) y_{q_j}$$

$$\text{for all } q_i \in Q^? \text{ and } (a, \sigma_a) \in \text{Steps}(q_i)$$

where y_{q_i} denotes the expected cost of reaching a state that satisfies ϕ from the state q_i. The problem admits a unique optimal solution and the actions that give rise to such optimal costs at states $q_i \in Q^?$ constitutes a stationary control policy. This optimal policy is unique.

B. Complex Probabilistic Computation Tree Logic Formulas

Complex PCTL formulas are those that consist of more than one $\mathcal{P}$- or $\mathcal{E}$-operator. This type of formulas include combinations of $\mathcal{P}$-operators, which allows more expressivity, hence, specifying complex tasks. This combination of formulas can be achieved by two methods: 1) nesting $\mathcal{P}$- and $\mathcal{E}$-operators and 2) using the Boolean operators. The methods of control synthesis for these formulas are discussed as follows.

1) Nesting $\mathcal{P}$- and $\mathcal{E}$-operators: Since each probabilistic and expected cost operator is a state formula itself, it is possible to combine these operators by nesting one inside another. Such a combination of these operators allows more expressivity in PCTL formulas. For instance, Specifications 2–4 in Section II, whose PCTL formulas are ϕ_2, ϕ_3, and ϕ_4 that are shown in (3)–(5), respectively, belong to this class of formulas.

It should be noted that we require all inner $\mathcal{P}$-operators to be of the form $\mathcal{P}_{\bowtie p}[\psi]$ as opposed to being $\mathcal{P}_{\max=?}[\psi]$. This is required because each nested probabilistic operator needs to identify a set of satisfying states. Generally, the nested formulas can be written in one of the following forms:

$$\phi = \mathcal{P}_{\max=?}[X\phi_R] \tag{6}$$

$$\phi = \mathcal{P}_{\max=?}[\phi_L \mathcal{U}^{\leq k} \phi_R] \tag{7}$$

$$\phi = \mathcal{P}_{\max=?}[\phi_L \mathcal{U} \phi_R] \tag{8}$$

where ϕ_R in (6) and at least one of ϕ_L and ϕ_R in (7) and (8) include a $\mathcal{P}$-operator. Subscripts L and R stand for to the left and right of the temporal operator, respectively.

Our method of producing a control strategy treats each probabilistic operator individually and proceeds as follows. First, we find the set of initial states Q_{ϕ_R}, from which ϕ_R is satisfied. A corresponding control policy μ_{ϕ_R} is also determined by applying the optimal algorithms that are given in Sections IV-A1a, A2a, and A3.

Next, ϕ_L is considered and the set of initial states Q_{ϕ_L} and the corresponding control policy μ_{ϕ_L} are determined. For ϕ_L, it is desired to find all the satisfying stationary policies. This is important for completeness of our solution. The PCTL formulas (7) and (8) require to reach a state in Q_{ϕ_R} only by going through Q_{ϕ_L} states. Thus, at Q_{ϕ_L} states, all and only the actions that satisfy ϕ_L are to be considered. As discussed in (see Section IV-A), however, finding all satisfying policies is only feasible for the temporal operator X because of the computational complexity of the other operators. For operators $\mathcal{U}^{\leq k}$ and $\mathcal{U}$ in ϕ_L, we use the stationary and optimal algorithms given in Sections IV-A2b and A3, respectively, to find μ_{ϕ_L}.

Then, we construct a new MDP $\mathcal{M}' \subseteq \mathcal{M}$ by eliminating the actions that are not allowed by μ_{ϕ_L} from states Q_{ϕ_L}. In other words, we remove all the action choices at states Q_{ϕ_L} except those allowed by μ_{ϕ_L} in $\mathcal{M}$. This step is performed to ensure the satisfaction of the path formula in ϕ_L. If this process results in states with no outgoing transition (blocking states), a self-transition is added to each of these states. This guarantees a new nonblocking MDP. In the last step, the optimal control algorithm is applied to the outermost $\mathcal{P}$-operator on the modified MDP $\mathcal{M}'$ to find the optimal control policy μ_ϕ and its corresponding probability value p_0 from the initial state q_0.

It should be noted that, by the nature of the PCTL formulas, the execution of the optimal policy μ_ϕ only guarantees satisfaction of a formula ϕ, which specifies that the system should reach a state in Q_{ϕ_R} through the states in Q_{ϕ_L}. Hence, the path formula that is specified in ϕ_R is not satisfied by μ_ϕ unless μ_{ϕ_R} is also executed. To ensure the execution of all the specified tasks in ϕ and ϕ_R, we construct a history-dependent control policy of the following form:

μ : *"Apply policy μ_ϕ until a state in Q_{ϕ_R} is reached. Then, apply policy μ_{ϕ_R}."*

For the same reason as state earlier, the returned probability value p_0 is the maximum probability of satisfying ϕ (reaching a state in Q_R through states of Q_L) under μ_ϕ from the initial state q_0. The probability of satisfying the path formula in ϕ_R from q_0 by executing policy μ cannot be found directly because it is not known: Which state in Q_R is reached first? However, since the probability of satisfying ϕ_R from each state in Q_R is available, a bound on the probability of satisfying ϕ and then ϕ_R from q_0 can be defined. The lower and upper limits of this bound are $p_0 p^{\min}_{\phi_R}$ and $p_0 p^{\max}_{\phi_R}$, respectively, where $p^{\min}_{\phi_R}$ and $p^{\max}_{\phi_R}$ denote the minimum and the maximum probabilities of satisfying ϕ_R from Q_{ϕ_R}, respectively.

Our approach for the control synthesis of nested formulas is quite general and allows for multiple levels of nested path formulas. This is because we treat each $\mathcal{P}$- and $\mathcal{E}$-operator individually. To find a control strategy for such formulas, a recursive calling of the aforementioned algorithm is applied starting with the innermost path operator.

To illustrate control synthesis of complex formulas with nested $\mathcal{P}$-operators, consider the MDP that is shown in Fig. 5 with the labels $L(q_0) = \{\mathbf{Init}\}$ for *Initial*, $L(q_2) = L(q_4) = \{\mathbf{U}\}$ for *Unsafe*, and $L(q_1) = \{\mathbf{D_2}\}$ and $L(q_5) = \{\mathbf{D_1}\}$ for *Destinations 1* and *2*, respectively, and the following complex PCTL formula:

$$\phi = \mathcal{P}_{\max=?}\left[\mathcal{P}_{\leq 0.3}\,[X\mathbf{U}]\ \mathcal{U}\ \left(\mathbf{D_1} \wedge \mathcal{P}_{\geq 0.4}[\Diamond^{\leq 3}\mathbf{D_2}]\right)\right].$$

In other words, ϕ means "Find the maximum probability and its corresponding policy to reach *Destination 1* by going only through the states from which the probability of converging to an *Unsafe* state is less than or equal to 0.3, and then with the probability greater than or equal to 0.4 reach *Destination 2* in less than three steps."

The algorithm proceeds with finding the initial states and control policies for $\phi_R = \left(\mathbf{D_1} \wedge \mathcal{P}_{\geq 0.4}[\Diamond^{\leq 3}\mathbf{D_2}]\right)$ and $\phi_L = \mathcal{P}_{\leq 0.3}\,[X\mathbf{U}]$. By applying the optimal control algorithm for "Bounded Until" (see Section IV-A2), we find the satisfying state, i.e., $Q_R = \{q_5\}$, with the control policy, i.e., $\mu_R(q_0) = a_1$ and $\mu_R(q_2) = \mu_R(q_4) = \mu_R(q_5) = a_2$ and the maximum probability of $p^{q_5}_{\mu_R} = 0.42$. The satisfying states of ϕ_L are $Q_L = \{q_0, q_1, q_2, q_4, q_5\}$, and all the satisfying actions are $\mu_L(q_0) = \mu_L(q_2) = \mu_L(q_5) = \{a_1\}$ and

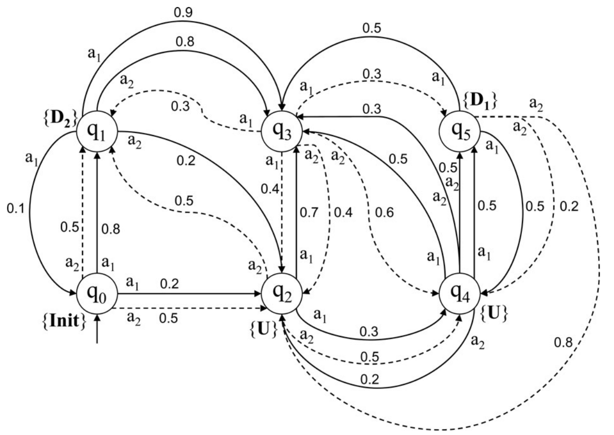

Fig. 5. MDP with six states and two actions per state. The labels are $L(q_0) = \{\mathbf{Init}\}$, $L(q_1) = \{\mathbf{D_2}\}$, $L(q_2) = L(q_4) = \{\mathbf{U}\}$, and $L(q_5) = \{\mathbf{D_1}\}$. The dashed edges correspond to the actions that do not satisfy $\mathcal{P}_{\leq 0.3}[X\,\mathbf{U}]$.

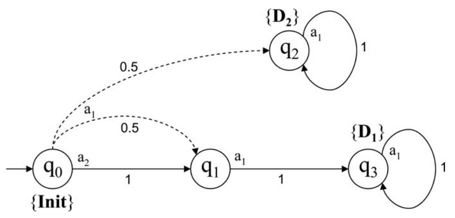

Fig. 6. MDP with four states. The dashed edges correspond to the actions that are eliminated from the new MDP $\mathcal{M}'$. This example illustrates the conservativeness of the PCTL control synthesis algorithm for complex formulas.

$\mu_L(q_1) = \mu_L(q_4) = \{a_1, a_2\}$. This is achieved by applying the algorithm that is given in Section IV-A1b. The actions that do not satisfy ϕ_L are shown as dashed edges in Fig. 5. Next, we eliminate these actions and construct a new MDP $\mathcal{M}'$. By performing the optimal control algorithm for "Until" (see Section IV-A3) on $\mathcal{M}'$, we find the maximum probability of satisfying ϕ from the initial state q_0 to be $p^{q_0}_{\mu_\phi} = 0.0574$ with the control policy, i.e., $\mu_\phi(q_0) = \mu_\phi(q_2) = a_1$ and $\mu_\phi(q_1) = \mu_\phi(q_4) = a_2$. The over all policy μ is "Apply μ_ϕ until q_5 is reached, and then apply μ_R". With this policy the probability of satisfying ϕ and then ϕ_R from q_0 is $(0.0574) \times (0.42) = 0.0241$.

Our solution for nested $\mathcal{P}$-operator formulas is conservative. As mentioned earlier, for completeness of the solution, we need to consider all the actions that satisfy ϕ_L at each state of Q_{ϕ_L}. However, because of computational complexity, we use the stationary and optimal algorithms for $\mathcal{U}^{\leq k}$ and $\mathcal{U}$ operators, respectively, which return only the optimal actions as opposed to all satisfying actions. Hence, our solution for the formulas whose ϕ_L include "bounded until" or "unbounded until" operators is not complete. In fact, for these formulas, the algorithm may return a suboptimal policy or may not find a solution at all while one might exist. This is demonstrated in the following example.

Consider the MDP in Fig. 6 and the formula

$$\phi = \mathcal{P}_{\max=?}[\mathcal{P}_{\geq 0.5}[\Diamond^{\leq 2}\mathbf{D_1}]\ \mathcal{U}\ \mathbf{D_2}].$$

Since ϕ_R does not include a $\mathcal{P}$-operator, we begin with applying the stationary algorithm for bounded until on $\phi_L = \mathcal{P}_{\geq 0.5}[\Diamond^{\leq 2}\mathbf{D_1}]$ and obtain $Q_L = \{q_0, q_1\}$ and $\mu_L(q_0) = a_2$ and

$\mu_L(q_1) = a_1$. Next, we construct a new MDP $\mathcal{M}'$ by eliminating action a_1 from the state q_0 and apply the optimal algorithm for $\mathcal{U}$. However, as shown in Fig. 6, there is no path to q_2 from q_0 in $\mathcal{M}'$. Thus, the algorithm will return zero as a solution to the maximum probability of achieving ϕ. Nevertheless, it is clear that a solution exists since q_0 satisfies ϕ_L with both actions a_1 and a_2. Hence, the optimal solution to this problem is 0.5 with the policy, i.e., $\mu(q_0) = \mu(q_1) = a_1$.

For the group of complex PCTL formulas, where ϕ_L does not include $\mathcal{U}^{\leq k}$ or $\mathcal{U}$, our solution is complete. This group of specifications includes many natural motion planning tasks that are useful in robotic applications. Examples include "Eventually reach *A* and then reach *B*, while always avoiding *C*" or "Eventually reach *A* through regions from which the probability of convergence to *C* is less than 0.3."

2) Combining $\mathcal{P}$-Operators by Boolean Operators: Another method to construct complex PCTL formulas is to combine $\mathcal{P}$-operators by the Boolean operators' conjunction "$\wedge$" and disjunction "$\vee$." These formulas are of the form, i.e., $\phi = \phi_1 \wedge \phi_2$ and $\phi = \phi_1 \vee \phi_2$, where both ϕ_1 and ϕ_2 include a path formula. In other words, ϕ_1 and ϕ_2 include $\mathcal{P}_{\bowtie p}[\psi]$.

To solve for formulas, i.e., $\phi = \phi_1 \wedge \phi_2$, first, we find the satisfying states Q_{ϕ_1} and Q_{ϕ_2} and their corresponding control policies μ_{ϕ_1} and μ_{ϕ_2}. In the case that ϕ_1 or ϕ_2 includes the temporal operator X ("next"), all the satisfying policies are considered. If ϕ_1 or ϕ_2 includes the temporal operator $\mathcal{U}^{\leq k}$ ("bounded until"), the stationary policy is considered. For the case of "until," the optimal control policy is determined. The solution exists only for the initial states, i.e., $Q_\phi = Q_{\phi_1} \cap Q_{\phi_2}$, given that μ_{ϕ_1} and μ_{ϕ_2} assign the same actions to these states. If Q_{ϕ_1} and Q_{ϕ_2} do not intersect, or the assigned actions for the states of Q_ϕ by μ_{ϕ_1} and μ_{ϕ_2} differ, no solution exists.

For the formulas, i.e., $\phi = \phi_1 \vee \phi_2$, we find the satisfying states Q_{ϕ_1} and Q_{ϕ_2} and their corresponding optimal control policies μ_{ϕ_1} and μ_{ϕ_2}. The policy that gives rise to the highest probability of satisfying the formula from the initial state q_0 is returned as a solution.

Similar to nested $\mathcal{P}$-operators, our method of finding a control policy for the complex specifications that include a combination of $\mathcal{P}$-operators with the Boolean operator conjunction "$\wedge$" is sound but not complete. In other words, the algorithm will only produce correct solutions but may fail to find a solution even though one exists. That is because we use stationary and optimal algorithms for the temporal operators "bounded until" and "until," respectively, which can only return the optimal actions regardless of the bound that is specified in the $\mathcal{P}$-operator.

C. Complexity

The overall time complexity for PCTL control synthesis for an MDP from a formula ϕ is linear in the size of the formula and the polynomial of the size of the model. To see this, we first define the size of a formula to be the sum of occurrences of $\mathcal{P}$- and $\mathcal{E}$-operators and the size of an MDP model to be $|\mathcal{M}| = \sum_{q_i \in Q} |\mathcal{A}(q_i)|$, which is the total number of actions available at all the states. In PCTL control synthesis, we are required to consider each operator of ϕ separately, thus the complexity becomes linear in the size of the formula. As mentioned earlier, the computational complexity of the "next" operator is only a matrix–vector multiplication followed by a 1-D search, while for the "bounded until," it is k matrix–vector multiplication and maximization/minimization operations. However, k is usually small in PCTL formulas. Hence, the most expensive cases are the "until" and "expected cost" operators, whose algorithms require solving linear optimization problems of the size of the model $|\mathcal{M}|$. This be can done in polynomial time by using, e.g., the ellipsoid methods [30].

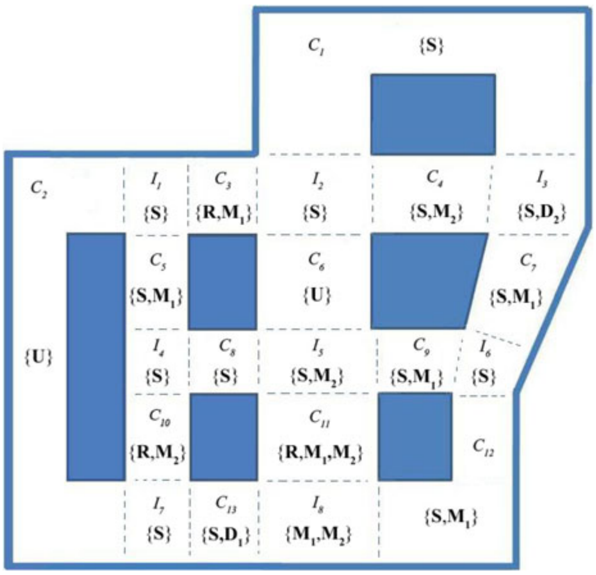

Fig. 7. Schematic representation of a larger indoor environment. It consists of 13 corridors ($C_1, \ldots, C_{13}$) and eight intersections ($I_1, \ldots, I_8$). The properties that are satisfied at the regions are shown between curly brackets inside the regions $\mathbf{S}$ = *Safe*, $\mathbf{R}$ = *Relatively safe*, $\mathbf{U}$ = *Unsafe*, $\mathbf{M_1}$ = *Medical supply type 1*, $\mathbf{M_2}$ = *Medical supply type 2*, $\mathbf{D_1}$ = *Destination 1*, and $\mathbf{D_2}$ = *Destination 2*.

V. Experiments and Simulations

To test the algorithms that are proposed in this paper, we deployed a robot from two simple and three complex PCTL specifications utilizing our RIDE shown in Fig. 1. The RIDE includes an experimental platform and a simulation tool. For the simple specifications, we performed simulations using the RIDE simulator with a small environment whose topology is schematically shown in Fig. 2. For the complex specifications, we considered a larger and more complex environment (see Fig. 7) in the RIDE simulator. To verify that the simulation results match with the real experiments, we also performed experimental trials on the small environment. We showed that the experimental and simulation results were statistically from the same distributions with 95% confidence by using chi-square and Fisher exact tests [31].

A. Experimental Platform and Simulation Tool

Our RIDE (see Fig. 1) consists of an environment with RFID tags and a robot that can navigate in the environment autonomously. The environment topology is easily reconfigurable and consists of corridors of various widths and lengths and intersections of several shapes and sizes. Each corridor and intersection is bounded by a line of RFID tags (white patches in Fig. 1 (left) and small ID numbers in Fig. 2) meant to trigger a transition event. The correct reading of such an event guarantees that the robot knows precisely its current region at any time. The mobile platform is an iRobot Create that is fitted with a Hokoyu URG-04LX laser range finder, APSX RW-210 RFID reader, and an MSI Wind U100-420US netbook. Commands are provided to the iCreate from Matlab using the iRobot Open Interface and the iCreate Matlab Toolbox [32]. The communications between the sensors, netbook, and robot occur through the use of USB connections.

The robot's motion is determined by specifying a forward velocity and angular velocity. At a given time, the robot implements one of the following four controllers (motion primitives): FollowRoad, GoRight, GoLeft, and GoStraight. Each of these controllers operates by obtaining data from the laser scanner and calculating a "target angle." The target angle represents the desired heading of the robot in order to execute the specified command, and this angle is translated into a proportional control law for angular velocity. The target angle in each case is found by utilizing two laser data points at certain angles that are relative to the robot's heading (different for each controller) and finding the midpoint between them. The target angle is then defined as the angle between the robot's heading and a line connecting the midpoint and the center of the robot. For each of these four controllers, the forward velocity control specified is based on the distance to the nearest obstacle in any direction. Therefore, as the robot approaches an obstacle, it will slow down. A cap is placed on the maximum velocity to keep the robot at reasonable speeds in more open areas of the environment. Each controller also provides for obstacle avoidance and emergency actions.

To test the robot control strategies, as well as to generate sample data necessary for the construction of the MDP model of the robot motion (see Section V-B), we use a RIDE simulator (see Fig. 8). The simulator was designed to resemble the physical setup very closely by emulating experimentally measured response times, sensing and control errors, and noise levels and distributions in the laser scanner readings. The configuration of the environment in the simulator is easily reconfigurable to capture changes in the topology of the experimental environment.

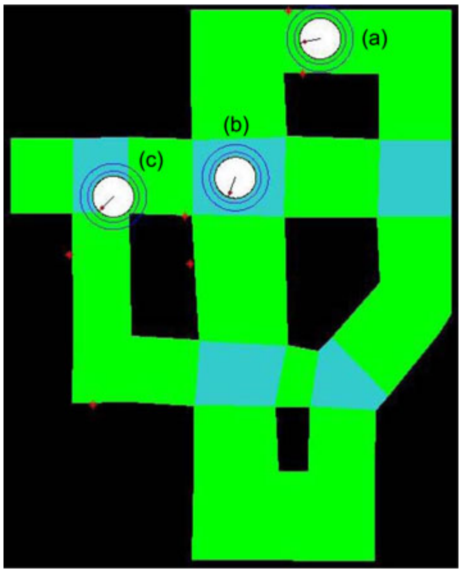

Fig. 8. Snapshots from the RIDE simulator. The robot is represented as a white disk. The arrow inside the white disk shows the robot's heading. The inner circle around the robot represents the "emergency" radius (if there is an obstacle within this zone, the emergency controller is used). The outer circle represents the radius within which the forward speed of the robot varies with the distance to the nearest obstacle. If there are no obstacles in this area, the robot moves at the maximum speed. The red dots are the laser readings that are used to define the target angle. (a) The robot centers itself on a stretch of road by using FollowRoad. (b) The robot applies GoRight in an intersection but fails to turn right because the laser readings do not properly identify the road on the right. (c) The robot applies GoLeft in an intersection and successfully turns left because the laser readings fall inside the road.

B. *Construction and Validation of the Markov Decision Process Model*

The small environment (see Fig. 2) consists of nine corridors and two four-way and three three-way intersections. The larger environment (see Fig. 7) has 13 corridors and two four-way and six three-way intersections. In the corridors only the controller FollowRoad is available. The controllers that are available at four-way intersections are GoLeft, GoRight, and GoStraight, while at the three-way intersections, only GoLeft and GoRight controllers are available.

In order to use an MDP to model the motion of the robot, we must ensure that the results of an action at a state depends only on the current state. It is intuitively clear from the environments in Figs. 2 and 7 that the orientation of the robot in a region depends on the region it came from. Therefore, the result of an action will depend on the previous region. To accommodate this dependence, we define the states of the MDP to be the ordered pairs of regions. For example, the state I_1–C_2 means that the robot was on C_1 and is now on I_2.

The set of actions that are available at a state of the MDP is the set of controllers that are available at the second region of the state. For example, when in the state C_1–I_2, only those actions from the region I_2 are allowed, which are GoLeft, GoRight, and GoStraight enabling transitions to the states I_2–C_4, I_2–C_3, and I_2–C_6, respectively. When designing the robot controllers, we also made sure that the robot did not get stuck in a region, i.e., the robot can only spend a finite amount of time in each region. Thus, the states are of the form intersection–corridor and corridor–intersection (states such as I_i–I_i or C_i–C_i do not exist). The resulting MDP for the small environment that is shown in Fig. 2 has 34 states, while the larger environment (see Fig. 7) includes 52 states.

To obtain transition probabilities, we performed a total of 500 simulations for each controller that is available in each MDP state. In each trial, the robot was initialized at the beginning of the first region of each state. If this region was a corridor, then the FollowRoad controller was applied until the system transitioned to the second region of the state. If the first region was an intersection then the controller most likely to transition the robot to the second region was applied. Once the second region was reached, one of the allowed actions was applied and the resulting transition was recorded. The results were then compiled into the transition probabilities.

The set of properties of the MDP was defined to be $\Pi = \{\mathbf{S}, \mathbf{R}, \mathbf{U}, \mathbf{M_1}, \mathbf{M_2}, \mathbf{D_1}, \mathbf{D_2}\}$, where $\mathbf{S} =$ *Safe*, $\mathbf{R} =$ *Relatively safe*, $\mathbf{U} =$ *Unsafe*, $\mathbf{M_1} =$ *Medical supply type 1*, $\mathbf{M_2} =$ *Medical supply type 2*, $\mathbf{D_1} =$ *Destination 1*, and $\mathbf{D_2} =$ *Destination 2*. Each state of the MDP was mapped to the set of properties that were satisfied at the second region of the state.

Finally, to capture the cost of applying an action at a state of the MDP, we assigned a nonnegative integer to each control primitive. In particular, we defined the costs of taking actions FollowRoad, GoRight, GoStraight, and GoLeft to be 0, 1, 2, and 3, respectively, which are independent of the state of the MDP.

C. Case Studies

As mentioned earlier, we considered two simple and three complex motion specifications in the environments that are shown in Figs. 2 and 7, respectively. We obtained the optimal control strategy and probability/expected cost value for each specification using our PCTL control synthesis algorithm. To verify these theoretical results, we performed simulation trials using the RIDE simulator. We also deployed the robot with the control strategies from the simple specifications in the RIDE's experimental platform to confirm both simulation and theoretical values.

The simple motions specifications that we considered in the small environment (see Fig. 2) are as follows.

Specification 5: Reach *Destination 1* by driving through either only *Safe* regions or through *Relatively safe* regions only if *Medical Supply 1* or *2* is available at such regions.

Specification 6: Reach *Destination 1* by driving through *Safe* or *Relatively safe* regions only.

The complex specifications that are given over the properties of the larger environment (see Fig. 7) are as follows.

Specification 7: Reach *Destination 1* by going through the regions from which the probability of converging to a *Relatively safe* region is less than 0.50 and always avoiding *Unsafe* regions.

Specification 8: Reach *Destination 1* by avoiding *Unsafe* regions and regions that are *Relatively safe* but do not have *Medical supply 1* and then with the probability greater than or equal to 0.50 reach *Destination 2* by driving through regions that are *Safe* or at which *Medical supply 2* is available.

Specification 9: Reach *Destination 1* and then with the probability 1 reach a region at which *Medical supplies 1* and *2* are available, while minimizing the total amount of the expected cost of the control actions.

Given that we are interested in the policy that produces the maximum probability of satisfying Specifications 5, 6, 7, and 8, they translate naturally to the PCTL formulas ϕ_5, ϕ_6, ϕ_7, and ϕ_8, respectively, where

$$\phi_5 : \mathcal{P}_{\max=?}\,[(\mathbf{S} \vee (\mathbf{R} \wedge (\mathbf{M_1} \vee \mathbf{M_2})))\,\mathcal{U}\,\mathbf{D_1}\,]$$

$$\phi_6 : \mathcal{P}_{\max=?}\,[(\mathbf{S} \vee \mathbf{R})\,\mathcal{U}\,\mathbf{D_1}\,]$$

$$\phi_7 : \mathcal{P}_{\max=?}\,[(\mathcal{P}_{<0.50}[X\,\mathbf{R}] \wedge \neg\mathbf{U})\,\mathcal{U}\,\mathbf{D_1}\,]$$

TABLE I
OBTAINED RESULTS FOR SPECIFICATIONS 5–9

Specification	Theoretical	Simulation	Experimental
ϕ_5	0.227	0.260	0.229
ϕ_6	0.674	0.642	0.629
ϕ_7	0.152	0.118	N/A
ϕ_8	0.259	0.244	N/A
ϕ_9	5.43	5.27	N/A

The values for ϕ_5, ϕ_6, ϕ_7, and ϕ_8 correspond to the probability of success, while the values for ϕ_9 indicate the total amount of the expected cost.

$$\phi_8 : \mathcal{P}_{\max=?}\,[\neg\mathbf{U} \wedge \neg(\mathbf{R} \wedge \neg\mathbf{M_1})\,\mathcal{U}\,(\mathbf{D_1} \wedge \mathcal{P}_{\geq 0.50}[(\mathbf{S} \vee \mathbf{M_2})\,\mathcal{U}\,\mathbf{D_2}])\,].$$

For Specification 9, we are interested in the policy that produces the minimum expected cost. Thus, it directly translates to the PCTL formula

$$\phi_9 : \mathcal{E}_{\min=?}\,[\mathbf{D_1} \wedge \mathcal{P}_{\geq 1}[\Diamond\,(\mathbf{M_1} \wedge \mathbf{M_2})\,]\,].$$

1) Theoretical Results: Assuming that the robot is initially at C_1 and is oriented toward I_2 in both environments, we computed the maximum probabilities to be 0.227 and 0.674 for Specifications 5 and 6, respectively, in the small environment. In the large environment, the maximum probability of satisfaction for Specification 7 was 0.152. For Specification 8, we found the maximum probability of reaching *Destination 1* to be 0.456, and the probability of reaching the first destination and then the second destination determined in this specification was 0.259. It should be noted that we were able to produce an exact number instead of a bound for the probability of satisfaction of the nested formula ϕ_8 because even though two states satisfy $\mathbf{D_1}$ (I_8–C_{13} and I_7–C_{13}), only one (I_8–C_{13}) is reachable from the initial state. Finally, the optimal policy that satisfies Specification 9 produces the minimum expected cost of 5.43. These theoretical values along with the simulation and experimental results are tabulated in Table I.

2) Simulation Results: To confirm the predicted probabilities and expected cost, we performed 500 simulations for each of the optimal control strategies. The simulations showed that the probabilities of satisfying ϕ_5 and ϕ_6 were 0.260 and 0.642, respectively, in the environment that is shown in Fig. 2. For Specification 7, the probability of satisfying ϕ_7 was 0.118 in the environment that is shown in Fig. 7. The rate of successful runs for the strategy that is obtained from ϕ_8 was 0.435 to reach *Destination 1* and 0.244 to reach *Destination 1* and then *Destination 2*. The average total cost of satisfying ϕ_9 was 5.27. It should be noted that the small discrepancy between the theoretical values and simulation results is likely because of remaining non-Markovian behavior of the transitions.

3) Experimental Results: In order to verify that the simulation results hold for the motion of the robot in the RIDE experimental platform, we carried out experimental runs in the small environment. As the first step, we made sure that the MDP that is obtained through the extensive simulation procedure was a good model for the actual motion of the robot in the experimental platform. For this, we randomly selected four transition probability distributions and experimentally determined the

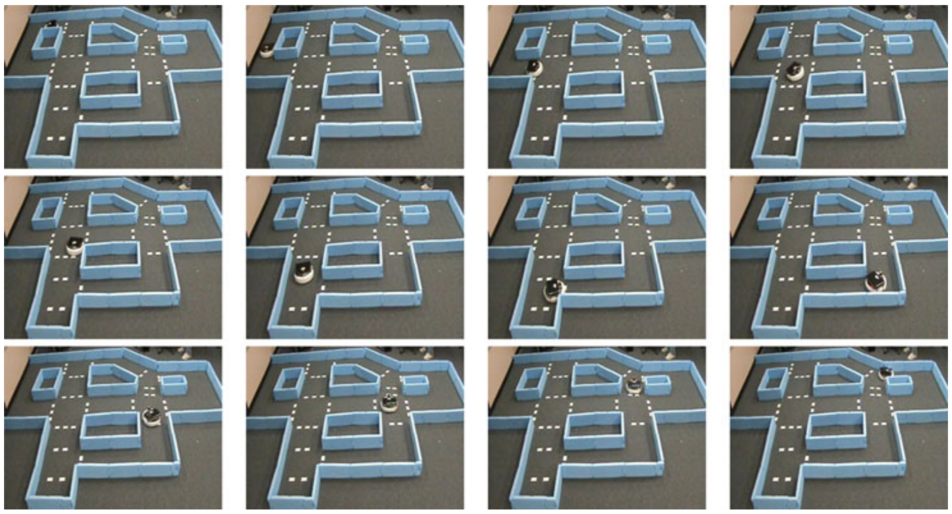
Fig. 9. Snapshots (to be read left-to-right and top-to-bottom) from a movie showing a robot motion that is produced by applying the control strategy maximizing the probability of satisfying Specification 2 (9).

transition probabilities. We then compared the simulated-based and experimental-based probabilities using either the Fisher exact test when only a few experimental trials were available or the chi-square test for homogeneity if there were a large number of experimental trials. These tests confirmed that the experimental data of four randomly selected transition probabilities were not significantly different from simulation results with a minimum certainty of 0.95. Next, we deployed the robot in the small environment of RIDE for strategies that are obtained from Specifications 5 and 6. This experiment was repeated 35 times for each control strategy. It was inferred that the probabilities of satisfying ϕ_5 and ϕ_6 were 0.229 and 0.629, respectively. By using the chi-square and Fisher's exact statistical tests, we concluded that the frequency of trials satisfying the specifications in the experiment matched the simulation data with a minimum certainty of 0.95.

Snapshots from a movie showing a motion of the robot that is produced by the control strategy maximizing the probability of satisfying Specification 6 are shown in Fig. 9. With reference to the notation that are given in Fig. 2, it can be seen that the robot follows the route $C_1 I_2 C_3 I_1 C_5 I_4 C_8 I_5 C_9$. It can be easily seen that this run satisfies Specification 2 (formula ϕ_2), because **S** or **R** is true until $\mathbf{D_1}$ is satisfied (at C_9). This movie, together with other movies of experimental and simulation trials that are obtained by applying the aforementioned control strategies, is available for download from [26].

D. Time Complexity

The control synthesis computations for the specifications that are given in formulas ϕ_8 and ϕ_9 were computationally the most expensive, i.e., because they required the solution of two linear programming problems on the large MDP (with 52 states and the size of 86). It took 0.053 and 0.059 s to synthesize control policies for ϕ_8 and ϕ_9, respectively, on a desktop computer with 2 Intel Xeon Quad-Core processors at 2.66 GHz with 3 GB RAM. However, the construction of the MDP through the Monte Carlo simulation took much longer (about 8 h). Considering the fact that the construction of the model is done offline and the computational complexity of the PCTL control synthesis algorithm is a polynomial in time, the computational framework that is described in this paper is reasonably scalable to more complex scenarios with larger environments. For example, this PCTL control synthesis method was utilized to solve a vehicle control problem in a hostile environment with more than 1000 states in its MDP model [33]. The computing time of the control synthesis from a nested PCTL formula of the form similar to ϕ_8 was 4 min on the same desktop computer.

VI. Conclusion

A computational framework has been presented for automatic deployment of a mobile robot from a temporal logic specification about properties of interest that are satisfied at the regions of a partitioned environment. The motion of the robot has been modeled as an MDP and the robot deployment problem has been mapped to the problem of generation of an MDP control strategy maximizing the probability of satisfying a PCTL formula. For the latter, a control synthesis algorithm has been presented that accepts all PCTL formulas. The method with simulation and experimental results has been illustrated in our RIDE.

This study can be extended to at least three directions in the future. One possible future work is to relax the assumption that the robot can determine its current region in the environment. The removal of this assumption results in a POMDP model of the motion of the robot in the environment. Thus, the problem becomes: How to generate a control strategy from a temporal logic formula for a POMDP? Another possible future direction is to extend the presented computational framework to a multiagent case, where a group of mobile robots can be deployed from a specification that is given as a temporal logic statement with probabilistic guarantees. This study can also be improved by developing an automated abstraction framework, which takes the dynamics and noise model of the robot and the map of the partitioned environment as input to construct an exact MDP model.

ACKNOWLEDGMENT

The authors would like to thank J. Wasniewski, K. Ryan, and B. Chang-Yun Hsu from the Boston University, Boston, MA, for their help with the construction of Robotic InDoor Environment (RIDE), the programming of Roomba, and the development of the RIDE simulator.

REFERENCES

[1] M. Lahijanian, J. Wasniewski, S. Andersson, and C. Belta, "Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees," in *Proc. IEEE Int. Conf. Robot. Autom.*, Anchorage, AK, 2010, pp. 3227–3232.

[2] M. Lahijanian, S. B. Andersson, and C. Belta, "Control of Markov decision processes from PCTL specifications," presented at the Amer. Control Conf., San Francisco, CA, 2011.

[3] S. M. LaVall, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[4] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Dept. Comput. Sci., Iowa State Univ., Ames, IA, Tech. Rep. 98–11, 1998..

[5] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[6] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Trans. Robot.*, vol. 26, no. 3, pp. 469–482, Jun. 2010.

[7] T. Bretl, "Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem," *Int. J. Robot. Res.*, vol. 25, no. 4, pp. 317–342, 2006.

[8] K. Hauser, T. Bretl, J. C. Latombe, K. Harada, and B. Wilcox, "Motion planning for legged robots on varied terrain," *Int. J. Robot. Res.*, vol. 27, no. 11–12, pp. 1325–1349, 2008.

[9] K. Hauser and J. C. Latombe, "Multi-modal motion planning in non-expansive spaces," in *Proc. Int. Workshop Algorithmic Found. Robot.*, Guanajuato, Mexico, 2008, pp. 615–630.

[10] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-robot motion planning: A timed automata approach," in *Proc. IEEE Int. Conf. Robot. Autom.*, New Orleans, LA, Apr. 2004, pp. 4417–4422.

[11] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Falsification of LTL safety properties in hybrid systems," in *Proc. 15th Int. Conf. Tools Algo. Construct. Anal. Syst.: Held Part Joint Eur. Conf. Theory Practice Softw.*, 2009, pp. 368–382.

[12] H. K. Gazit, G. Fainekos, and G. J. Pappas, "Where's Waldo? sensor-based temporal logic motion planning," in *Proc. IEEE Conf. Robot. Autom.*, Rome, Italy, 2007, pp. 3116–3121.

[13] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. Automat. Control*, vol. 53, no. 1, pp. 287–297, Feb. 2008.

[14] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proc. Int. Conf. Hybrid Syst., Comput. Control*, 2010, pp. 101–110.

[15] E. M. M. Clarke, D. Peled, and O. Grumberg, *Model Checking*. Cambridge, MA: MIT Press, 1999.

[16] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive(1) designs," in *Proc. Int. Conf. Verification, Model Checking Abstract Interpretation*, Charleston, SC, 2006, pp. 364–380.

[17] M Kloetzer and C. Belta, "Dealing with non-determinism in symbolic control," in *Hybrid Systems: Computation and Control: 11th International Workshop* (Lecture Notes in Computer Science), M. Egerstedt and B. Mishra, Eds. Berlin, Germany: Springer, 2008, pp. 287–300.

[18] D. Bertsekas, *Dynamic Programming and Optimal Control*. vol. 1, Belmont, MA: Athena Scientific, 1995.

[19] J. Pineau, G. Gordon, and S. Thrun, "Anytime point-based approximations for large POMDPs," *J. Artif. Intell. Res.*, vol. 27, pp. 335–380, 2006.

[20] N. L. Zhang and W. Zhang, "Speeding up the convergence of value iteration in partially observable Markov decision processes," *J. Artif. Intell. Res.*, vol. 14, pp. 29–51, 2001.

[21] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time Markov chains," *IEEE Trans. Softw. Eng.*, vol. 29, no. 6, pp. 524–541, 2003.

[22] C. Baier, "On algorithmic verification methods for probabilistic systems," Ph.D. dissertation, Fakultät für Mathematik & Informatik, Universität Mannheim, 1998.

[23] L. de Alfaro, *Model Checking of Probabilistic and Nondeterministic Systems*. New York: Springer-Verlag, 1995, pp. 499–513.

[24] R. Alterovitz, T. Siméon, and K. Goldberg, "The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty," presented at the Robot., Sci. Syst., Conf., Atlanta, GA, 2007.

[25] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," *Int. J. Softw. Tools Technol. Transf.*, vol. 6, no. 2, pp. 128–142, 2004.

[26] Robotic indoor environment (RIDE). (2010). [Online]. Available: hyness.bu.edu/ride/

[27] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* (CRM Monograph Series 23), P. Panangaden and F. van Breugel, Eds., Providence, RI, 2004.

[28] L. de Alfaro, "Formal verification of probabilistic systems," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, 1997.

[29] D. P. Bertsekas and J. N. Tsitsiklis, "An analysis of stochastic shortest path problems," *Math. Oper. Res.*, vol. 16, pp. 580–595, 1991.

[30] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*. Belmont, MA: Athena Scientific, 1997.

[31] A. Trujillo-Ortiz, R. Hernandez-Walls, A. Castro-Perez, L. Rodriguez-Cardozo, N. Ramos-Delgado, and R. Garcia-Sanchez. (2004). Fisherextest: Fisher's exact probability test. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/5957

[32] J. M. Esposito and O. Barton. (2008). Matlab toolbox for the iRobot create. [Online]. Available: www.usna.edu/Users/weapsys/esposito/roomba.matlab/

[33] I. Cizelj, X. C. Ding, M. Lahijanian, A. Pinto, and C. Belta, "Probabilistically safe vehicle control in a hostile environment," presented at the Int. Fed. Automat. Control 18th World Congr., Milan, Italy, 2011.

Morteza Lahijanian (S'11) received the B.S. degree in bioengineering from University of California, Berkeley, in 2004 and the M.S. degree in mechanical engineering from Boston University, Boston, MA, in 2009, where he is currently working toward the Ph.D. degree in mechanical engineering.

His research interests include dynamics and control theory with applications in robotics and systems biology, motion planning with discrete abstraction, formal synthesis, and hybrid systems.

Sean B. Andersson (M'03) received the B.S. degree in engineering and applied physics from Cornell University, Ithaca, NY, in 1994, the M.S. degree in mechanical engineering from Stanford University, Stanford, CA, in 1995, and the Ph.D. degree in electrical and computer engineering from the University of Maryland, College Park, in 2003.

He was a Project Engineer with AlliedSignal Aerospace in 1995, a Senior Controls Engineer with Aerovironment during 1996–1998, and a Lecturer in applied mathematics with Harvard University during 2003–2005. He is currently an Assistant Professor of mechanical engineering and of systems engineering with Boston University, Boston, MA. His research interests include systems and control theory with applications in scanning probe microscopy, dynamics in molecular systems, and robotics.

Dr. Andersson received the National Science Foundation CAREER award in 2009.

Calin Belta (M'03) received the B.S. and M.Sc. degrees in control and computer science from the Technical University of Iasi, Iasi, Romania, and the M.Sc. and Ph.D. degrees in mechanical engineering from the University of Pennsylvania, Philadelphia.

He is currently an Associate Professor with Boston University, Boston MA. His research interests include dynamics and control, motion planning, robotics, and systems biology.

Dr. Belta is an Associate Editor of the *SIAM Journal on Control and Optimization* and of the IEEE Robotics and Automation Society and Control Systems Society Conference Editorial Boards. He received the Air Force Office of Scientific Research Young Investigator Award in 2008 and the National Science Foundation CAREER Award in 2005.