Incremental Synthesis of Control Policies for Heterogeneous Multi-Agent Systems with Linear Temporal Logic Specifications

Tichakorn Wongpiromsarn, Alphan Ulusoy, Calin Belta, Emilio Frazzoli and Daniela Rus

Abstract— **We consider automatic synthesis of control policies for non-independent, heterogeneous multi-agent systems with the objective of maximizing the probability of satisfying a given specification. The specification is expressed as a formula in linear temporal logic. The agents are modeled by Markov decision processes with a common set of actions. These actions, however, may or may not affect the behaviors of all the agents. To alleviate the well-known state explosion problem, an incremental approach is proposed where only a small subset of agents is incorporated in the synthesis procedure initially and more agents are successively added until the limitations on computational resources are reached. The proposed algorithm runs in an anytime fashion, where the probability of satisfying the specification increases as the algorithm progresses.**

I. Introduction

Multi-agent systems have played an increasingly important role in robotics research as robots often have to interact with one another as well as other uncontrollable and partially controllable dynamic agents, including humans. In addition, the robots may have to perform more complex tasks than target tracking, obstacle avoidance and going from one point to another as typically studied in the robotics literature.

Linear temporal logic (LTL) has been demonstrated to be a powerful language for specifying complex robotics tasks [1]–[5]. Furthermore, it has been shown that control policies that ensure or maximize the probability for the robots to satisfy a specification expressed as a formula in LTL can be automatically synthesized based on exhaustive exploration of the state space [6]–[13].

For systems that contain multiple agents, the size of the state space grows exponentially with the number of agents. As a result, the control policy synthesis problem becomes more computationally complex as more agents are incorporated in the synthesis procedure. Consider, as an example, the problem where an autonomous vehicle needs to go through a pedestrian crossing while there are multiple pedestrians who are already at or approaching the crossing. The state space of the complete system (i.e., the vehicle and all the pedestrians) grows exponentially with the number of the pedestrians. As a result, given a limited budget of computational resources, solving the control policy synthesis problem with respect to LTL specifications may not be feasible when there are a large number of pedestrians. This issue, commonly known as the state explosion problem, has been a key obstacle that limits the application of existing control policy synthesis approaches to relatively small problems.

To alleviate the state explosion problem, distributed synthesis where a global specification of the system is decomposed into local specifications that can then be used to synthesize control policies for the individual agents has been proposed [14], [15]. The approach in [14], however, is limited to the case where the global specification is *traced-closed* whereas the approach in [15] requires manual decomposition of the global specification and can be conservative. In addition, distributed synthesis is not suitable for problems that involve a large number of environment agents over which the system does not have control.

In [16], [17], we proposed an anytime algorithm for synthesizing a control policy for a robot interacting with multiple independent environment agents with the objective of maximizing the probability for the robot to satisfy a given specification. The main idea was to progressively compute a sequence of control policies, by taking into account only a small subset of the environment agents initially and successively adding more agents to the synthesis procedure in each iteration until the computational resource constraints were exceeded. Incremental construction of various objects that needed to be computed during the synthesis procedure was proposed to avoid unnecessary computation by exploiting the objects computed in the previous iteration. However, we only considered a specific case where the robot was modeled by a Markov decision process whereas each environment agent was modeled by a Markov chain. Hence, an action of the robot could not affect the behavior of the environment agents. Furthermore, the specification was limited to the co-safety fragment of LTL, which only admits tasks that can be satisfied in finite time. Hence, tasks such as persistent surveillance could not be specified. Incremental verification and synthesis for other types of systems and specifications can be found, e.g., in [18]–[20].

In this paper, we generalize our previous work [16], [17] by considering more general systems and more general specifications. The main contribution of this paper is twofold. First, we consider full LTL, rather than only its co-safety fragment; thus, allowing more expressive specifications. Second, as opposed to [16], [17], which only considered a robot interacting with multiple independent environment agents, we consider more general multi-agent systems, including those where robots interact with each other as well as envi-

This work was partially supported by Singapore National Research Foundation, SMART IRG on Future Urban Mobility, NSF CNS-1016213, ONR-MURI N00014-09-1051, ONR MURI N00014-10-10952, NSF CNS-0834260 and NSF CNS-1035588.

T. Wongpiromsarn is with the Singapore-MIT Alliance for Research and Technology, Singapore 117543, Singapore. nok@smart.mit.edu

A. Ulusoy and C. Belta are with Boston University, Boston, MA, USA alphan@bu.edu, cbelta@bu.edu

E. Frazzoli and D. Rus are with the Massachusetts Institute of Technology, Cambridge, MA, USA frazzoli@mit.edu, rus@csail.mit.edu

ronment agents. In addition, agents may not be completely independent. We model each agent by a Markov decision process. Two case studies are considered, illustrating that we are able to obtain an optimal solution faster than existing approaches. Another key advantage of our incremental approach is its anytime nature. For large problems that cannot be fully solved, our incremental approach may still provide a reasonable, potentially sub-optimal solution.

II. Preliminaries

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

A. Linear Temporal Logic and Automata

Linear temporal logic is a powerful specification language for precisely expressing a wide range of properties of systems. An LTL formula is built up from a set Π of atomic propositions, the logic connectives $\neg$, $\vee$, $\wedge$ and $\Longrightarrow$ and the temporal modal operators $\bigcirc$ ("next"), $\Box$ ("always"), $\Diamond$ ("eventually") and $\mathcal{U}$ ("until"). LTL formulas are interpreted on infinite strings $\sigma = \sigma_0\sigma_1\sigma_2\ldots$ where $\sigma_i \in 2^\Pi$ for all $i \geq 0$. Such infinite strings are referred to as "words." The satisfaction relation is denoted by $\models$, i.e., for a word σ and an LTL formula φ, we write $\sigma \models \varphi$ iff σ satisfies φ. We refer the reader to [1]–[3] for more details on LTL.

Given propositional formulas p_1 and p_2, examples of widely used LTL formulas include (i) a safety formula $\Box p_1$ (read as "always p_1"), which simply asserts that property p_1 remains invariantly true throughout an execution, (ii) 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) and (iii) a progress formula $\Box\Diamond p_1$ (read as "always eventually p_1"), which states that the property p holds infinitely often in an execution.

As we will see later, there is a tight relationship between LTL and finite state automata that will be exploited in control policy synthesis.

Definition 1: A *deterministic Rabin automaton* (DRA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_{init}, Acc)$ where (a) Q is a finite set of states, (b) Σ is a finite set called alphabet, (c) $\delta : Q \times \Sigma \to Q$ is a transition function, (d) $q_{init} \in Q$ is the initial state, and (e) $Acc \subseteq 2^Q \times 2^Q$ 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 σ in a DRA $\mathcal{A} = (Q, \Sigma, \delta, q_{init}, Acc)$ is an infinite sequence of states $q_0q_1\ldots$ such that $q_0 = q_{init}$ and $q_i \xrightarrow{\sigma_i} q_{i+1}$ for all $i \geq 0$. A run is *accepting* if there exists a pair $(H, K) \in Acc$ such that there exists $n \geq 0$ such that for all $m \geq n$, $q_m \notin H$ and there exist infinitely many $n \geq 0$ such that $q_n \in K$.

A string $\sigma \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there is an accepting run of σ in $\mathcal{A}$. The language *accepted* by $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of all accepted strings of $\mathcal{A}$. It can be shown that for any LTL formula φ over Π, there exists a DRA $\mathcal{A}$ with alphabet $\Sigma = 2^\Pi$ that accepts all and only words over Π that satisfy φ, i.e., $\mathcal{L}(\mathcal{A}) = \{\sigma \in (2^\Pi)^\omega \mid \sigma \models \varphi\}$. Such $\mathcal{A}$ can be automatically constructed using existing tools [21].

B. Systems and Control Policies

We consider the case where each component of the system is modeled by a Markov decision process, defined as follows.

Definition 2: A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, \Pi, L)$ where (a) S is a countable set of states, (b) Act is a countable set of actions, (c) $\mathbf{P} : S \times Act \times S \to [0, 1]$ is the transition probability function such that for any $s \in S$ and $\alpha \in Act$, $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\}$, (d) $\iota_{init} : S \to [0, 1]$ is the initial state distribution satisfying $\sum_{s \in S} \iota_{init}(s) = 1$, (e) Π is a set of atomic propositions, and (f) $L : S \to 2^\Pi$ is a labeling function. $\mathcal{M}$ is called *finite* if S, Act and Π are finite. A *valid initial state* of $\mathcal{M}$ is a state $s \in S$ such that $\iota_{init}(s) > 0$. An action α is *enabled* in state s if and only if $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$. We let $Act(s)$ denote the set of enabled actions in s.

Assuming that all the components of the system make a transition simultaneously, the complete system can be constructed based on the synchronous parallel composition of all the components. Synchronous parallel composition of MDPs can be defined based on the definition of handshaking composition [3] of transition systems as follows.

Definition 3: Let $\mathcal{M}_1 = (S_1, Act, \mathbf{P}_1, \iota_{init,1}, \Pi_1, L_1)$ and $\mathcal{M}_2 = (S_2, Act, \mathbf{P}_2, \iota_{init,2}, \Pi_2, L_2)$ be Markov decision processes. Their synchronous parallel composition, denoted by $\mathcal{M}_1||\mathcal{M}_2$, is the MDP $\mathcal{M} = (S_1 \times S_2, Act, \mathbf{P}, \iota_{init}, \Pi_1 \cup \Pi_2, L)$ where (a) for each $s_1, s_1' \in S_1$, $s_2, s_2' \in S_2$ and $\alpha \in Act$, $\mathbf{P}(\langle s_1, s_2\rangle, \alpha, \langle s_1', s_2'\rangle) = \mathbf{P}_1(s_1, \alpha, s_1')\mathbf{P}_2(s_2, \alpha, s_2')$, (b) for each $s_1 \in S_1$ and $s_2 \in S_2$, $\iota_{init}(\langle s_1, s_2\rangle) = \iota_{init,1}(s_1)\iota_{init,2}(s_2)$, and (c) for each $s_1 \in S_1$ and $s_2 \in S_2$, $L(\langle s_1, s_2\rangle) = L(s_1) \cup L(s_2)$.

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 $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, \Pi, L)$ be a Markov decision process. A *control policy* for $\mathcal{M}$ is a function $\mathcal{C} : S^+ \to Act$ such that $\mathcal{C}(s_0s_1 \ldots s_n) \in Act(s_n)$ for all $s_0s_1 \ldots s_n \in S^+$.

Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, \Pi, L)$ be an MDP and $\mathcal{C} : S^+ \to Act$ be a control policy for $\mathcal{M}$. Given an initial state s_0 of $\mathcal{M}$ such that $\iota_{init}(s_0) > 0$, an infinite sequence $r^{\mathcal{C}}_{\mathcal{M}} = s_0s_1 \ldots$ on $\mathcal{M}$ generated under policy $\mathcal{C}$ is called a *path* on $\mathcal{M}$ if $\mathbf{P}(s_i, \mathcal{C}(s_0s_1 \ldots s_i), s_{i+1}) > 0$ for all i. The subsequence $s_0s_1 \ldots s_n$ where $n \geq 0$ is the *prefix* of length n of $r^{\mathcal{C}}_{\mathcal{M}}$. We define $Paths^{\mathcal{C}}_{\mathcal{M}}$ and $FPaths^{\mathcal{C}}_{\mathcal{M}}$ as the set of all infinite paths of $\mathcal{M}$ under policy $\mathcal{C}$ and their finite prefixes, respectively, starting from any state s_0 with $\iota_{init}(s_0) > 0$. For $s_0s_1 \ldots s_n \in FPaths^{\mathcal{C}}_{\mathcal{M}}$, we let $Paths^{\mathcal{C}}_{\mathcal{M}}(s_0s_1 \ldots s_n)$ denote the set of all paths in $Paths^{\mathcal{C}}_{\mathcal{M}}$ with the prefix $s_0s_1 \ldots s_n$. It can be shown [3] that there exists a unique probability measure $Pr^{\mathcal{C}}_{\mathcal{M}}$ on the $\sigma-$algebra associated with $\mathcal{M}$ under policy $\mathcal{C}$ where for any $s_0s_1 \ldots s_n \in FPaths^{\mathcal{C}}_{\mathcal{M}}$,

$$\begin{aligned}&\mathrm{Pr}^{\mathcal{C}}_{\mathcal{M}}\{Paths^{\mathcal{C}}_{\mathcal{M}}(s_0s_1 \ldots s_n)\} = \\ &\qquad \iota_{init}(s_0) \textstyle\prod_{0 \leq i < n} \mathbf{P}(s_i, \mathcal{C}(s_0s_1 \ldots s_i), s_{i+1}).\end{aligned}$$

Given an LTL formula φ, one can show that the set $\{s_0 s_1 \ldots \in Paths^{\mathcal{C}}_{\mathcal{M}} \mid L(s_0)L(s_1)\ldots \models \varphi\}$ is measurable [3]. The probability for $\mathcal{M}$ to satisfy φ under policy $\mathcal{C}$ is then defined as

$$\Pr^{\mathcal{C}}_{\mathcal{M}}(\varphi) = \Pr^{\mathcal{C}}_{\mathcal{M}}\{s_0 s_1 \ldots \in Paths^{\mathcal{C}}_{\mathcal{M}} \mid L(s_0)L(s_1)\ldots \models \varphi\}.$$

For each state $s \in S$, we let $\mathcal{M}^s = (S, Act, \mathbf{P}, \iota^s_{init}, \Pi, L)$ where $\iota^s_{init}(t) = 1$ if $s = t$ and $\iota^s_{init}(t) = 0$ otherwise. We define $\Pr^{\mathcal{C}}_{\mathcal{M}}(s \models \varphi) = \Pr^{\mathcal{C}}_{\mathcal{M}^s}(\varphi)$ as the probability for $\mathcal{M}$ to satisfy φ under policy $\mathcal{C}$, starting from s.

A control policy essentially resolves all the nondeterministic choices in an MDP and induces a Markov chain $\mathcal{M}_{\mathcal{C}}$ that formalizes the behavior of $\mathcal{M}$ under control policy $\mathcal{C}$ [3]. In general, $\mathcal{M}_{\mathcal{C}}$ contains all the states in S^+ and hence may not be finite even though $\mathcal{M}$ is finite. However, for a special case where $\mathcal{C}$ is a memoryless or a finite memory control policy, it can be shown that $\mathcal{M}_{\mathcal{C}}$ 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 N (possibly heterogeneous) agents. Agent $i \in \{1, \ldots, N\}$ is modeled by a finite Markov decision process $\mathcal{M}_i = (S_i, Act, \mathbf{P}_i, \iota_{init,i}, \Pi_i, L_i)$ where Act is the set of all the control actions that are available to the system.

We assume that at any time instance, the state of the system, which incorporates the state of all the agents, can be precisely observed. In addition, we assume that all the agents $\mathcal{M}_1, \ldots, \mathcal{M}_N$ make a transition simultaneously, i.e., each of them makes a transition at every time step and their interaction can be captured by the synchronous parallel composition (see Definition 3). To ensure well-definedness of the complete system, we assume that for all $i \in \{1, \ldots, N\}$ and $s \in S_i$, $Act(s) = Act$. Without this assumption, there may exist a state of the complete system in which no valid control action can be applied.

Control Policy Synthesis Problem: Given a system model described by $\mathcal{M}_1, \ldots, \mathcal{M}_N$ and an LTL formula φ over $\Pi_1 \cup \ldots \cup \Pi_N$, we want to automatically synthesize an optimal control policy $\mathcal{C}$ that maximizes the probability $\Pr^{\mathcal{C}}_{\mathcal{M}}(\varphi)$ for the system $\mathcal{M} = \mathcal{M}_1 || \ldots || \mathcal{M}_N$ to satisfy φ.

Example 1: Consider a mobility-on-demand system [22] where a vehicle needs to transfer passengers between $N-1$ stations as shown in Figure 1. The vehicle is modeled by an MDP $\mathcal{M}_N = (\{st_1, \ldots, st_{N-1}\}, Act, \mathbf{P}_N, \iota_{init,N}, \Pi_N, L_N)$ where st_i corresponds to station i. Each action $\alpha \in Act$ corresponds to a path the vehicle can take. Π_N is defined as the set of labels of all the stations. In this case, the labeling function L_N essentially maps each station to its label. Station i is modeled by an MDP $\mathcal{M}_i = (S_i, Act, \mathbf{P}_i, \iota_{init,i}, \Pi_i, L_i)$ whose state $s \in S_i$ captures the number of passengers waiting at the station. Each proposition in Π_i describes the abstract state of the station, e.g., the level of crowdedness. Note

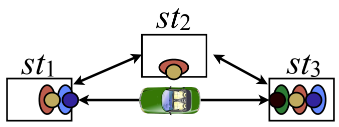


Fig. 1. The mobility-on-demand example.

that the number of passengers waiting at each station depends on both the passenger arrival rate and the action taken by the vehicle. In this case, a control policy for the system needs to ensure that station 1 (e.g., the most important station) always remains uncrowded whereas all the other stations cannot be crowded for more than 1 time step consecutively. In addition, we may also require that the vehicle visits each station infinitely often. In this case, the specification φ is given by $\varphi = \Box\neg c_1 \wedge \bigwedge_{i\in\{2,\ldots,N-1\}} \Box(c_i \implies \bigcirc\neg c_i) \wedge \bigwedge_{i\in\{1,\ldots,N-1\}} \Box\Diamond st_i$ where for each $i \in \{1, \ldots, N\}$, $c_i \in \Pi_i$ is a label that indicates that station i is crowded.

Remark 1: Control actions in Act do not necessarily affect the behaviors of all the agents. For example, there may exist an agent i with $\mathbf{P}_i(s, \alpha, s') = \mathbf{P}_i(s, \alpha', s')$ for all $\alpha, \alpha' \in Act$ and $s, s' \in S_i$, i.e., the transition probability between each pair of states remains the same for all actions. In [16], [17], such an agent is referred to as an "environment" agent. In this paper, however, we do not distinguish between an "environment" agent and a "robot." In addition, an action may affect multiple agents; hence, the agents may not be completely independent.

IV. MDP Control from LTL Specifications

A typical approach to solve the control policy synthesis problem defined in Section III is to first compute the composition of all the system components to obtain the complete system. Based on Definition 3, the complete system can be modeled by the MDP $\mathcal{M}_1 || \ldots || \mathcal{M}_N$. We denote this MDP by $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, \Pi, L)$. Next, we apply existing results for synthesizing control policies for MDP from LTL specifications [3], [12]. Roughly, this involves constructing the product MDP and extracting an optimal control policy for the product MDP as briefly described next. We refer the reader to [3], [12] for more details

A. Construction of Product MDP

Let $\mathcal{A}_\varphi = (Q, 2^\Pi, \delta, q_{init}, Acc)$ be a DRA that accepts all and only words over Π that satisfy φ. The first step for synthesizing a control policy for $\mathcal{M}$ from specification φ is to obtain a finite MDP $\mathcal{M}_p = (S_p, Act_p, \mathbf{P}_p, \iota_{p,init}, Q, L_p)$ as the product of $\mathcal{M}$ and $\mathcal{A}_\varphi$, defined as follows.

Definition 5: Let $\mathcal{M} = (S, Act, \mathbf{P}, \iota_{init}, \Pi, L)$ be an MDP and let $\mathcal{A} = (Q, 2^\Pi, \delta, q_{init}, Acc)$ be a DRA. The product of $\mathcal{M}$ and $\mathcal{A}$ is the MDP $\mathcal{M}_p = \mathcal{M} \otimes \mathcal{A}$ defined by $\mathcal{M}_p = (S_p, Act, \mathbf{P}_p, \iota_{p,init}, \Pi, L_p)$ where $S_p = S \times Q$ and $L_p(\langle s, q\rangle) = L(s)$. $\mathbf{P}_p$ is defined as

$$\mathbf{P}_p(\langle s,q\rangle, \alpha, \langle s',q'\rangle) = \begin{cases} \tilde{\mathbf{P}}_p(\langle s,q\rangle, \alpha, \langle s',q'\rangle) & \\ & \text{if } q' = \delta(q, L(s')) \\ 0 & \text{otherwise} \end{cases},$$

where $\tilde{\mathbf{P}}_p(\langle s,q\rangle, \alpha, \langle s',q'\rangle) = \mathbf{P}(s, \alpha, s')$. For the rest of the paper, we refer to $\tilde{\mathbf{P}}_p : S_p \times Act \times S_p \to [0,1]$ as the *intermediate transition probability function* for $\mathcal{M}_p$. Finally,

$$\iota_{p,init}(\langle s,q\rangle) = \begin{cases} \tilde{\iota}_{p,init}(\langle s,q\rangle) & \text{if } q = \delta(q_{init}, L(s)) \\ 0 & \text{otherwise} \end{cases},$$

where $\tilde{\iota}_{p,init}(\langle s,q\rangle) = \iota_{init}(s)$. For the rest of the paper, we refer to $\tilde{\iota}_{p,init} : S_p \to [0,1]$ as the *intermediate initial state distribution* for $\mathcal{M}_p$.

Consider a path $r^{\mathcal{C}_p}_{\mathcal{M}_p} = \langle s_0, q_0\rangle\langle s_1, q_1\rangle \ldots$ on $\mathcal{M}_p$ generated under some control policy $\mathcal{C}_p$. We say that $r^{\mathcal{C}_p}_{\mathcal{M}_p}$ is *accepting* if and only if $q_{init}q_0q_1\ldots$ is an accepting run on $\mathcal{A}_\varphi$, i.e., 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 $q_n \in K$.

Stepping through the above definition shows that given a path $r^{\mathcal{C}_p}_{\mathcal{M}_p} = \langle s_0, q_0\rangle\langle s_1, q_1\rangle \ldots$ on $\mathcal{M}_p$ generated under some control policy $\mathcal{C}_p$, the corresponding path $s_0s_1\ldots$ on $\mathcal{M}$ generates a word $L(s_0)L(s_1)\ldots$ that satisfies φ if and only if $r^{\mathcal{C}_p}_{\mathcal{M}_p}$ is accepting. Hence, each accepting path of $\mathcal{M}_p$ uniquely corresponds to a path of $\mathcal{M}$ whose word satisfies φ. In addition, a control policy $\mathcal{C}_p$ on $\mathcal{M}_p$ induces the corresponding control policy $\mathcal{C}$ on $\mathcal{M}$. The details for generating $\mathcal{C}$ from $\mathcal{C}_p$ can be found, e.g. in [3], [12].

B. Control Policy Synthesis for Product MDP

From probabilistic verification [3], it has been shown that the maximum probability for $\mathcal{M}$ to satisfy φ is equivalent to the maximum probability of reaching a certain set of states of $\mathcal{M}_p$ known as *accepting maximal end components* (AMECs). Before defining AMECs, we first provide the definition of *maximal end components* (MECs) as follows.

Definition 6: An *end component* of an MDP $\mathcal{M} = (S, Act, \mathbf{P}, s_{p,init}, \Pi, L)$ is a pair (T,A) where $\emptyset \neq T \subseteq S$ and $A : T \to 2^{Act}$ such that (1) $\emptyset \neq A(s) \subseteq Act(s)$ for all $s \in T$, (2) the directed graph induced by (T,A) is strongly connected, and (3) for all $s \in T$ and $\alpha \in A(s)$, $\{t \in S_p \mid \mathbf{P}_p(s,\alpha,t) > 0\} \subseteq T$.

An end component (T,A) is *maximal* if there is no end component $(T', A') \neq (T,A)$ such that $T \subseteq T'$ and $A(s) \subseteq A'(s)$ for all $s \in T$. Based on iterative computations of strongly connected components (SCCs), the set of all MECs can be computed with the worst-case time complexity that is quadratic in the size of the MDP [3].

Definition 7: Let $\mathcal{M} = (S, Act, \mathbf{P}, s_{p,init}, \Pi, L)$ be an MDP and $\mathcal{A} = (Q, 2^\Pi, \delta, q_{init}, Acc)$ be a DRA. An *accepting maximal end component* of $\mathcal{M}_p = \mathcal{M} \otimes \mathcal{A}$ is a maximal end component (T,A) of $\mathcal{M}_p$ such that for some $(H,K) \in Acc$, $H \cap T = \emptyset$ and $K \cap T \neq \emptyset$.

An AMEC (T,A) of $\mathcal{M}_p$ has the important property that starting from any state in T, there exists a finite memory control policy for $\mathcal{M}_p$ to keep the state within T forever while visiting all states in T infinitely often with probability 1. Hence, the maximum probability for $\mathcal{M}$ to satisfy φ is equivalent to the maximum probability of reaching S_G, where S_G contains all the states in the AMECs of $\mathcal{M}_p$. Thus, our next step is to compute the maximum probability of reaching S_G. For the rest of the paper, we use LTL-like notations to describe events in MDPs. In particular, $\Diamond S_G$ denotes 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, $x_s = \sup_{\mathcal{C}_p} \Pr^{\mathcal{C}_p}_{\mathcal{M}_p}(s \models \Diamond S_G)$. Well-known techniques for computing the probability x_s for each $s \in S_p$ include linear programming (LP) and value iteration [3]. LP-based techniques yield an exact solution but they typically do 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 $\lim_{k\to\infty} x_s^{(k)} = x_s$ for all $s \in S_p$.

SCC-based value iteration (SCC-VI) has been proposed to speed up value iteration [23], [24]. It can be shown that given all the SCCs $C_1^{\mathcal{M}_p}, \ldots, C_m^{\mathcal{M}_p}$ of $\mathcal{M}_p$, an order $\mathbb{O}^{\mathcal{M}_p}$ among $C_1^{\mathcal{M}_p}, \ldots, C_m^{\mathcal{M}_p}$ can be generated such that the probability values of states in $C_j^{\mathcal{M}_p}$ that appears after $C_i^{\mathcal{M}_p}$ in $\mathbb{O}^{\mathcal{M}_p}$ cannot affect the probability values of states in $C_i^{\mathcal{M}_p}$. Hence, we can apply value iteration to each SCC separately, according to the order in $\mathbb{O}^{\mathcal{M}_p}$. When processing $C_i^{\mathcal{M}_p}$, we exploit the order in $\mathbb{O}^{\mathcal{M}_p}$ and existing values of x_t for each state $t \in S_p \setminus C_i^{\mathcal{M}_p}$ that is an immediate successor of states in $C_i^{\mathcal{M}_p}$ to determine the set of $s \in C_i^{\mathcal{M}_p}$ where $x_s^{(k+1)}$ needs to be updated from $x_s^{(k)}$. Processing of SCC $C_i^{\mathcal{M}_p}$ terminates when all $x_s^{(k)}$, $s \in C_i^{\mathcal{M}_p}$ converges. We refer the reader to [23], [24] for more details.

Once the vector $(x_s)_{s\in S_p}$ is computed, a finite memory control policy $\mathcal{C}_p$ for $\mathcal{M}_p$ that maximizes the probability for $\mathcal{M}$ to satisfy φ can be constructed as follows. First, consider the case when $\mathcal{M}_p$ is in state $s \in S_G$. In this case, s belongs to some AMEC (T,A) and the policy $\mathcal{C}_p$ selects an action $\alpha \in A(s)$ such that all actions in $A(s)$ are scheduled infinitely often. (For example, $\mathcal{C}_p$ may select the action for s according to a round-robin policy.) Next, consider the case when $\mathcal{M}_p$ is in state $s \in S_p \setminus S_G$. In this case, $\mathcal{C}_p$ picks an action to ensure that $\Pr^{\mathcal{C}_p}_{\mathcal{M}}(s \models \Diamond S_G) = x_s$ can be achieved. If $x_s = 0$, an action in $Act_p(s)$ can be chosen arbitrarily. Otherwise, $\mathcal{C}_p$ picks an action $\alpha \in Act_p^{max}(s)$ such that $\mathbf{P}_p(s,\alpha,t) > 0$ 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 $\alpha \in Act_p^{max}(s)$, $x_s = \sum_{t\in S_p} \mathbf{P}(s,\alpha,t)x_t$ 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. Incremental Computation of Control Policies

The automatic synthesis procedure described in the previous section suffers from the state explosion problem as the composition $\mathcal{M}_1 || \ldots || \mathcal{M}_N$, whose size grows exponentially with N, needs to be constructed, leading to an exponential blow up of the state space. In this section, we propose an incremental synthesis approach, where we progressively compute a sequence of control policies, by taking into account only a small subset of agents initially and successively adding more agents to the synthesis procedure in each iteration until we hit the computational resource constraints. Hence, even though the complete synthesis problem cannot be solved due to the computational resource limitation, we can still obtain a reasonably good control policy.

Let $\mathbf{N} = \{1, \ldots, N\}$. Initially, we consider a small subset $\mathbf{N}_0 \subset \mathbf{N}$ of agents. For each agent $i \notin \mathbf{N}_0$, we compute the ordered set *SCC*$(\mathcal{M}_i)$ of all SCCs and the set *MEC*$(\mathcal{M}_i)$ of all MECs of $\mathcal{M}_i$ and construct a simplified model $\tilde{\mathcal{M}}_i$ that essentially assumes that agent i is stationary under all the control actions (i.e., we take into account the presence of agent i but do not consider its full model). Formally, $\tilde{\mathcal{M}}_i = (\{s^{\tilde{\mathcal{M}}_i}\}, Act, \tilde{\mathbf{P}}_i, \tilde{\iota}_{init,i}, \Pi_i, \tilde{L}_i)$ where $s^{\tilde{\mathcal{M}}_i} \in S_i$ can be chosen arbitrarily, $\tilde{\mathbf{P}}_i(s^{\tilde{\mathcal{M}}_i}, \alpha, s^{\tilde{\mathcal{M}}_i}) = 1$, for all $\alpha \in Act$, $\tilde{\iota}_{init,i}(s^{\tilde{\mathcal{M}}_i}) = 1$ and $\tilde{L}_i(s^{\tilde{\mathcal{M}}_i}) = L_i(s^{\tilde{\mathcal{M}}_i})$. Note that the choice of $s^{\tilde{\mathcal{M}}_i} \in S_i$ may affect the performance of our incremental synthesis algorithm; hence, it should be chosen such that it is the most likely state of $\mathcal{M}_i$.

The composition $\mathcal{M}^{\mathbf{N}_0} = \Big|\Big|_{i \in \mathbf{N}_0} \mathcal{M}_i \;||\; \Big|\Big|_{i \in \mathbf{N}\setminus\mathbf{N}_0} \tilde{\mathcal{M}}_i$ is then constructed. Note that since $\tilde{\mathcal{M}}_i$ is typically smaller than $\mathcal{M}_i$, $\mathcal{M}^{\mathbf{N}_0}$ is typically much smaller than $\mathcal{M}_1 || \ldots || \mathcal{M}_N$. We identify the ordered set *SCC*$(\mathcal{M}^{\mathbf{N}_0})$ of all SCCs and the set *MEC*$(\mathcal{M}^{\mathbf{N}_0})$ of all MECs of $\mathcal{M}^{\mathbf{N}_0}$. Then, following the steps for synthesizing a control policy described in Section IV, we construct $\mathcal{M}_p^{\mathbf{N}_0} = \mathcal{M}^{\mathbf{N}_0} \otimes \mathcal{A}_\varphi$. We also store the intermediate transition probability function $\tilde{\mathbf{P}}_p^{\mathbf{N}_0}$ and the intermediate initial state distribution $\tilde{\iota}_{p,init}^{\mathbf{N}_0}$ of $\mathcal{M}_p^{\mathbf{N}_0}$. At the end of the initialization period (i.e., iteration 0), we obtain a control policy $\mathcal{C}^{\mathbf{N}_0}$ that maximizes the probability for $\mathcal{M}^{\mathbf{N}_0}$ to satisfy φ.

Our algorithm then successively adds more full models of the rest of the agents to the synthesis procedure at each iteration. In iteration $k > 0$, we consider the set $\mathbf{N}_k = \mathbf{N}_{k-1} \cup \{l\}$ of agents for some $l \in \mathbf{N} \setminus \mathbf{N}_{k-1}$ and let $\mathcal{M}^{\mathbf{N}_k} = \Big|\Big|_{i \in \mathbf{N}_k} \mathcal{M}_i \;||\; \Big|\Big|_{i \in \mathbf{N}\setminus\mathbf{N}_k} \tilde{\mathcal{M}}_i$. Following the similar procedure as in iteration 0, we identify the ordered set *SCC*$(\mathcal{M}^{\mathbf{N}_k})$ of all SCCs and the set *MEC*$(\mathcal{M}^{\mathbf{N}_k})$ of all MECs of $\mathcal{M}^{\mathbf{N}_k}$ and construct $\mathcal{M}_p^{\mathbf{N}_k} = \mathcal{M}^{\mathbf{N}_k} \otimes \mathcal{A}_\varphi$, while storing its intermediate transition probability function $\tilde{\mathbf{P}}_p^{\mathbf{N}_k}$ and its intermediate initial state distribution $\tilde{\iota}_{p,init}^{\mathbf{N}_k}$. Finally, we obtain a control policy $\mathcal{C}^{\mathbf{N}_k}$ that maximizes the probability for $\mathcal{M}^{\mathbf{N}_k}$ to satisfy φ. Note that $\mathcal{C}^{\mathbf{N}_k}$ can be applied to the complete system $\mathcal{M}$ by considering $\mathcal{C}^{\mathbf{N}_k}$ to be only a function of states of agents in $\mathbf{N}_k$. The probability for the complete system $\mathcal{M}$ to satisfy φ under $\mathcal{C}^{\mathbf{N}_k}$ can then be efficiently computed using probabilistic verification [3].

The process outlined in the previous paragraph terminates at iteration $\bar{k}$ when one of the following conditions hold: (1) $\mathbf{N}_{\bar{k}} = \mathbf{N}$, (2) the complete system under policy $\mathcal{C}^{\mathbf{N}_{\bar{k}}}$ satisfies φ with probability 1, or (3) the computational resource constraints are exceeded. This idea also appears in [16]. However, in [16], we consider the case where only one agent (i.e., the robot) is modeled by a finite MDP while the other agents (i.e., the environment agents) are modeled by finite Markov chains. The approach proposed in [16] then allows us to incrementally compute the product MDP and a control strategy that maximizes the probability of reaching a given set S_G of "goal" states, while avoiding unnecessary computation by exploiting the objects computed in the previous iteration. Since only co-safety formulas φ were considered, the set S_G in [16] could be easily identified from the set of *final* states of the deterministic finite automaton that recognizes the good prefixes of φ. In this paper, we consider the case where all the agents are modeled by finite MDPs and we allow any specification that can be expressed in LTL (not necessarily the co-safety fragment). The computation of the AMECs of $\mathcal{M}_p^{\mathbf{N}_k}$ for each iteration k is therefore necessary to identify S_G.

Consider an arbitrary iteration $k \geq 0$ and let l be the index such that $\mathbf{N}_{k+1} = \mathbf{N}_k \cup \{l\}$. First, note that a state s_p of $\mathcal{M}_p^{\mathbf{N}_k}$ is of the form $s_p = \langle s, q \rangle$ where $s \in S_0 \times S_1 \times \ldots \times S_N$ and $q \in Q$. For $s = \langle s_0, s_1, \ldots, s_N \rangle \in S_0 \times S_1 \times \ldots \times S_N$, $i \in \{0, \ldots, N\}$ and $r \in S_i$, we define $s|_{i \leftarrow r} \triangleq \langle s_0, \ldots, s_{i-1}, r, s_{i+1}, \ldots, s_N \rangle$, i.e., $s|_{i \leftarrow r}$ is obtained by replacing the ith element of s by r. The following lemma shows that $\mathcal{M}_p^{\mathbf{N}_{k+1}}$, $\tilde{\mathbf{P}}_p^{\mathbf{N}_{k+1}}$ and $\tilde{\iota}_{p,init}^{\mathbf{N}_{k+1}}$ can be incrementally constructed from $\mathcal{M}_p^{\mathbf{N}_k}$, $\tilde{\mathbf{P}}_p^{\mathbf{N}_k}$ and $\tilde{\iota}_{p,init}^{\mathbf{N}_k}$. This allows us to avoid computing $\mathcal{M}^{\mathbf{N}_{k+1}}$.

Lemma 1: Let $\mathcal{M}_p^{\mathbf{N}_k} = (S_p^{\mathbf{N}_k}, Act, \mathbf{P}_p^{\mathbf{N}_k}, \iota_{p,init}^{\mathbf{N}_k}, \Pi, L_p^{\mathbf{N}_k})$. Suppose $\Pi_i \cap \Pi_j = \emptyset$ for all $i \neq j$. Then, $\mathcal{M}_p^{\mathbf{N}_{k+1}} = (S_p^{\mathbf{N}_{k+1}}, Act, \mathbf{P}_p^{\mathbf{N}_{k+1}}, \iota_{p,init}^{\mathbf{N}_{k+1}}, \Pi, L_p^{\mathbf{N}_{k+1}})$ where $S_p^{\mathbf{N}_{k+1}} = \{\langle s|_{l \leftarrow r}, q \rangle \mid \langle s, q \rangle \in S_p^{\mathbf{N}_k}$ and $r \in S_l\}$ and for any $s = \langle s_0, \ldots, s_N \rangle, s' = \langle s'_0, \ldots, s'_N \rangle \in S_0 \times \ldots S_N$ and $q, q' \in Q$,

- $\mathbf{P}_p^{\mathbf{N}_{k+1}}(\langle s, q \rangle, \alpha, \langle s', q' \rangle) = 0$ if $q' \neq \delta(q, L_p^{\mathbf{N}_{k+1}}(\langle s', q' \rangle))$. Otherwise, $\mathbf{P}_p^{\mathbf{N}_{k+1}}(\langle s, q \rangle, \alpha, \langle s', q' \rangle) = \tilde{\mathbf{P}}_p^{\mathbf{N}_{k+1}}(\langle s, q \rangle, \alpha, \langle s', q' \rangle)$, where the intermediate transition probability function is given by $\tilde{\mathbf{P}}_p^{\mathbf{N}_{k+1}}(\langle s, q \rangle, \alpha, \langle s', q' \rangle) = \mathbf{P}_l(s_l, \alpha, s'_l)$ $\tilde{\mathbf{P}}_p^{\mathbf{N}_k}(\langle \tilde{s}, q \rangle, \alpha, \langle \tilde{s}', q' \rangle)$ for any $\langle \tilde{s}, q \rangle, \langle \tilde{s}', q' \rangle \in S_p^{\mathbf{N}_k}$ such that $\tilde{s}|_{l \leftarrow s_l} = s$ and $\tilde{s}'|_{l \leftarrow s'_l} = s'$,
- $\iota_{p,init}^{\mathbf{N}_{k+1}}(\langle s, q \rangle) = 0$ if $q \neq \delta(q_{init}, L_p^{\mathbf{N}_{k+1}}(\langle s, q \rangle))$. Otherwise, $\iota_{p,init}^{\mathbf{N}_{k+1}}(\langle s, q \rangle) = \tilde{\iota}_{p,init}^{\mathbf{N}_{k+1}}(\langle s, q \rangle)$, where the intermediate initial state distribution is given by $\tilde{\iota}_{p,init}^{\mathbf{N}_{k+1}}(\langle s, q \rangle) = \iota_{init,l}(s_l) \tilde{\iota}_{p,init}^{\mathbf{N}_k}(\langle \tilde{s}, q \rangle)$ for any $\langle \tilde{s}, q \rangle \in S_p^{\mathbf{N}_k}$ such that $\tilde{s}|_{l \leftarrow s_l} = s$, and
- $L_p^{\mathbf{N}_{k+1}}(\langle s, q \rangle) = \big(L_p^{\mathbf{N}_k}(\langle \tilde{s}, q \rangle) \setminus L_l(\tilde{s}_l)\big) \cup L_l(s_l)$ for any $\langle \tilde{s}, q \rangle \in S_p^{\mathbf{N}_k}$ such that $\tilde{s}|_{l \leftarrow s_l} = s$.

The following two lemmas then show how the set of MECs of $\mathcal{M}_p^{\mathbf{N}_{k+1}}$ can be incrementally constructed.

Lemma 2: For any maximal end component $(T_p^{\mathbf{N}_k}, A_p^{\mathbf{N}_k})$ of $\mathcal{M}_p^{\mathbf{N}_k}$, $T_p^{\mathbf{N}_k} \subseteq \{\langle s, q \rangle \mid s \in T^{\mathbf{N}_k}, q \in Q\}$ for some maximal end component $(T^{\mathbf{N}_k}, A^{\mathbf{N}_k})$ of $\mathcal{M}^{\mathbf{N}_k}$.

Proof: Since $(T_p^{\mathbf{N}_k}, A_p^{\mathbf{N}_k})$ is an end component of $\mathcal{M}_p^{\mathbf{N}_k}$, it can be easily checked from the definition of product MDP that $(T^{\mathbf{N}_k}, A^{\mathbf{N}_k})$ is an end component of $\mathcal{M}^{\mathbf{N}_k}$ where $T^{\mathbf{N}_k} = \{s \mid \langle s, q \rangle \in T_p^{\mathbf{N}_k}\}$ and for each $s \in T^{\mathbf{N}_k}$, $A^{\mathbf{N}_k}(s) = \bigcup_{q \text{ s.t. } \langle s,q \rangle \in T_p^{\mathbf{N}_k}} A_p^{\mathbf{N}_k}(\langle s, q \rangle)$. Hence, we can conclude that for any $\langle s, q \rangle \in T_p^{\mathbf{N}_k}$, $s \in T^{\mathbf{N}_k}$ and $q \in Q$ where $(T^{\mathbf{N}_k}, A^{\mathbf{N}_k})$ is the end component of $\mathcal{M}^{\mathbf{N}_k}$ associated with the end component $(T_p^{\mathbf{N}_k}, A_p^{\mathbf{N}_k})$ of $\mathcal{M}_p^{\mathbf{N}_k}$. ■

Lemma 3: For any maximal end component $(T^{\mathbf{N}_{k+1}}, A^{\mathbf{N}_{k+1}})$ of $\mathcal{M}^{\mathbf{N}_{k+1}}$, $T^{\mathbf{N}_{k+1}} \subseteq \{s|_{l \leftarrow r} \mid s \in T^{\mathbf{N}_k}, r \in T^l\}$ for some maximal end component $(T^{\mathbf{N}_k}, A^{\mathbf{N}_k})$ of $\mathcal{M}^{\mathbf{N}_k}$ and some maximal end component (T^l, A^l) of $\mathcal{M}_l$.

Proof: Since $(T^{\mathbf{N}_{k+1}}, A^{\mathbf{N}_{k+1}})$ is an end component of $\mathcal{M}^{\mathbf{N}_{k+1}}$, from the definition of parallel composition of MDPs, it can be shown that $(T^{\mathbf{N}_k}, A^{\mathbf{N}_k})$ is an end component of $\mathcal{M}^{\mathbf{N}_k}$ and (T^l, A^l) is an end component of $\mathcal{M}_l$ where $T^{\mathbf{N}_k} = \{s|_{l \leftarrow s^{\tilde{\mathcal{M}}_l}} \mid s \in T^{\mathbf{N}_{k+1}}\}$ $T^l = \{s_l \mid \langle s_0, \ldots, s_N \rangle \in T^{\mathbf{N}_{k+1}}\}$, for each $s \in T^{\mathbf{N}_k}$, $A^{\mathbf{N}_k}(s) = \bigcup_{\tilde{s} \text{ s.t. } \tilde{s}|_{l \leftarrow s^{\tilde{\mathcal{M}}_l}} = s} A^{\mathbf{N}_{k+1}}(\tilde{s})$ and for each $s \in T^l$, $A^l(s_l) = \bigcup_{\tilde{s} \text{ s.t. } \tilde{s}_l = s_l} A^{\mathbf{N}_{k+1}}(\tilde{s})$. Hence, we can conclude that for any $s \in T^{\mathbf{N}_{k+1}}$, $s|_{l \leftarrow s^{\tilde{\mathcal{M}}_l}} \in T^{\mathbf{N}_k}$ and $s_l \in T^l$ where $T^{\mathbf{N}_k}, A^{\mathbf{N}_k}$ and (T^l, A^l) are maximal end components of $\mathcal{M}^{\mathbf{N}_k}$ and $\mathcal{M}^l$, respectively. ∎

Based on Lemma 1–3, Algorithm 1 summarizes our approach for incremental computation of control policies. It relies on construction of AMECs of a subset of states of the product MDP. Algorithm 2 adapts the AMECs computation algorithm [3] to allow such partial construction of AMECs. The correctness of Algorithm 1 can be directly derived from Lemma 1–3 and the correctness of the AMECs computation algorithm [3] as formally stated below.

Proposition 1: Algorithm 1 correctly returns $\mathcal{M}_p^{\mathbf{N}_{k+1}}$, $\tilde{\mathbf{P}}_p^{\mathbf{N}_{k+1}}$, $\tilde{\iota}_{p,init}^{\mathbf{N}_{k+1}}$, $SCC(\mathcal{M}^{\mathbf{N}_{k+1}})$, $MEC(\mathcal{M}^{\mathbf{N}_{k+1}})$ and $\mathcal{C}^{\mathbf{N}_{k+1}}$.

Algorithm 1: Incremental control policy synthesis

```
1  Input: $\mathcal{M}_p^{\mathbf{N}_k}$, $\tilde{\mathbf{P}}_p^{\mathbf{N}_k}$, $\tilde{\iota}_{p,init}^{\mathbf{N}_k}$, $SCC(\mathcal{M}^{\mathbf{N}_k})$, $MEC(\mathcal{M}^{\mathbf{N}_k})$,
   $\mathcal{A}_\varphi$, $\mathcal{M}_l$, $SCC(\mathcal{M}_l)$, $MEC(\mathcal{M}_l)$
2  Output: $\mathcal{M}_p^{\mathbf{N}_{k+1}}$, $\tilde{\mathbf{P}}_p^{\mathbf{N}_{k+1}}$, $\tilde{\iota}_{p,init}^{\mathbf{N}_{k+1}}$, $SCC(\mathcal{M}^{\mathbf{N}_{k+1}})$,
   $MEC(\mathcal{M}^{\mathbf{N}_{k+1}})$, $\mathcal{C}^{\mathbf{N}_{k+1}}$
3  Compute $\mathcal{M}_p^{\mathbf{N}_{k+1}}$, $\tilde{\mathbf{P}}_p^{\mathbf{N}_{k+1}}$ and $\tilde{\iota}_{p,init}^{\mathbf{N}_{k+1}}$ from $\mathcal{M}_p^{\mathbf{N}_k}$, $\tilde{\mathbf{P}}_p^{\mathbf{N}_k}$
   and $\tilde{\iota}_{p,init}^{\mathbf{N}_k}$ as described in Lemma 1
4  $MEC(\mathcal{M}^{\mathbf{N}_{k+1}}) := \emptyset$
5  foreach $T^{\mathbf{N}_k} \in MEC(\mathcal{M}^{\mathbf{N}_k})$ do
6      foreach $T^l \in MEC(\mathcal{M}_l)$ do
7          Add $\{s|_{l \leftarrow r} \mid s \in T^{\mathbf{N}_k}, r \in T^l\}$ to $MEC(\mathcal{M}^{\mathbf{N}_{k+1}})$
8  goal := $\emptyset$
9  foreach $T^{\mathbf{N}_{k+1}} \in MEC(\mathcal{M}^{\mathbf{N}_{k+1}})$ do
10     Compute $AMEC(\mathcal{M}_p^{\mathbf{N}_{k+1}}, T^{\mathbf{N}_{k+1}}, \mathcal{A}_\varphi)$ using
       Algorithm 2
11     foreach $T_p^{\mathbf{N}_{k+1}} \in AMEC(\mathcal{M}_p^{\mathbf{N}_{k+1}}, T^{\mathbf{N}_{k+1}}, \mathcal{A}_\varphi)$ do
12         foreach $s \in T_p^{\mathbf{N}_{k+1}}$ do
13             Add $s$ to goal
14 Incrementally compute $SCC(\mathcal{M}^{\mathbf{N}_{k+1}})$ and $\mathcal{C}^{\mathbf{N}_{k+1}}$ using
   the algorithm presented in [16], which takes
   $SCC(\mathcal{M}^{\mathbf{N}_k})$, $SCC(\mathcal{M}_l)$, $\mathcal{M}_p^{\mathbf{N}_{k+1}}$ and goal as input
15 Return: $\mathcal{M}_p^{\mathbf{N}_{k+1}}$, $\tilde{\mathbf{P}}_p^{\mathbf{N}_{k+1}}$, $\tilde{\iota}_{p,init}^{\mathbf{N}_{k+1}}$, $SCC(\mathcal{M}^{\mathbf{N}_{k+1}})$,
   $MEC(\mathcal{M}^{\mathbf{N}_{k+1}})$, $\mathcal{C}^{\mathbf{N}_{k+1}}$
```

VI. Experimental Results

Case Study 1: We revisit the mobility-on-demand problem described in Example 1. Suppose there are 3 stations. The models of the vehicle and the stations are shown in Figure 2. The DRA $\mathcal{A}_\varphi$ automatically generated using ltl2dstar [21] contains 37 states.

We apply the LP-based and SCC-based value iteration techniques described in Section IV to synthesize a control policy that maximizes the probability that the complete system $\mathcal{M} = \mathcal{M}_1 || \mathcal{M}_2 || \ldots || \mathcal{M}_4$ satisfies φ. In this case, the product MDP $\mathcal{M}_p = \mathcal{M} \otimes \mathcal{A}_\varphi$ contains 2997 states. The time required for each step of computation is summarized in Table I[1]. Both approaches generate a control policy that ensures that $\mathcal{M}$ satisfies φ with probability 1. The total computation times for the LP-based and SCC-based value iteration techniques are 64.55 and 74.67 seconds, respectively.

Next, we apply the incremental technique described in Section V. We let $\mathbf{N}_0 = \{4\}$, i.e., the full model of the vehicle is considered whereas for the stations, only the simplified models are used during initialization. For each $i \in \{1, 2, 3\}$,

Algorithm 2: Computing AMECs of a sub-MDP

```
1  Input: $\mathcal{A}_\varphi = (Q, 2^\Pi, \delta, q_{init}, Acc)$,
   $\mathcal{M}_p^{\mathbf{N}_{k+1}} = (S_p^{\mathbf{N}_{k+1}}, Act, \mathbf{P}_p^{\mathbf{N}_{k+1}}, \iota_{p,init}^{\mathbf{N}_{k+1}}, \Pi, L_p^{\mathbf{N}_{k+1}})$,
   $T^{\mathbf{N}_{k+1}} \subseteq S^{\mathbf{N}_{k+1}}$
2  Output: $AMEC(\mathcal{M}_p^{\mathbf{N}_{k+1}}, T^{\mathbf{N}_{k+1}}, \mathcal{A}_\varphi)$
3  foreach $s \in S_p^{\mathbf{N}_{k+1}}$ do
4      $A(s) := Act(s)$
5  AMEC := $\emptyset$; AMEC$_{new}$ := $\{T^{\mathbf{N}_{k+1}} \times Q\}$
6  while AMEC $\neq$ AMEC$_{new}$ do
7      AMEC := AMEC$_{new}$; AMEC$_{new}$ := $\emptyset$
8      foreach $T \in AMEC$ do
9          $R := \emptyset$
10         Compute the set SCC of nontrivial SCCs of
           $(T, Act, \mathbf{P}_p^{\mathbf{N}_{k+1}}, \iota_{p,init}^{\mathbf{N}_{k+1}}, \Pi, L_p^{\mathbf{N}_{k+1}})$
11         foreach $C \in$ SCC do
12             foreach $s \in C$ do
13                 $A(s) := \{\alpha \in A(s) \mid \forall t \text{ s.t. } \mathbf{P}_p^{\mathbf{N}_{k+1}}(s, \alpha, t) >$
                   $0, t \in C\}$
14                 if $A(s) = \emptyset$ then
15                     Add $s$ to $R$
16         while $R \neq \emptyset$ do
17             Pick an arbitrary $s \in R$
18             Remove $s$ from $R$ and $T$
19             foreach $t \in T, \beta \in A(t)$ s.t. $\mathbf{P}_p^{\mathbf{N}_{k+1}}(t, \beta, s) > 0$
               do
20                 Remove $\beta$ from $A(t)$
21                 if $A(t) = \emptyset$ then
22                     Add $t$ to $R$
23         foreach $C \in$ SCC do
24             if $\exists (H, K) \in Acc$ s.t. $T \cap C \cap H = \emptyset$ and
               $T \cap C \cap K \neq \emptyset$ then
25                 Add $T \cap C$ to AMEC$_{new}$
26 Return: AMEC
```

[1]The computation time is implementation dependent. In this section, we compare the computation times of various approaches under the same implementation of $\mathcal{M}_p$, SCCs, AMECs, probability vector and control policy computation in Python. These computation times can be improved by using a more efficient implementation.

Technique	$\mathcal{M}_p$	SCCs	AMECs	Prob vector	Control policy	Total
Single pass, LP	26.5	-	23.22	14.79	0.04	64.55
Single pass, SCC-VI	26.5	24.68	23.22	0.24	0.04	74.67
Incremental	0.14	3×10^{-5}	0.04	0.001	0.0005	0.18

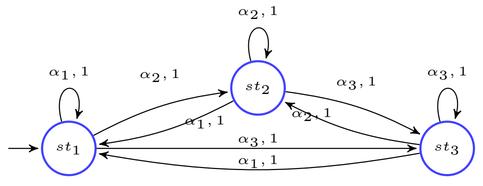
TABLE I

MOBILITY-ON-DEMAND: TIME REQUIRED (IN SECONDS) FOR COMPUTING VARIOUS OBJECTS DURING CONTROL POLICY SYNTHESIS.

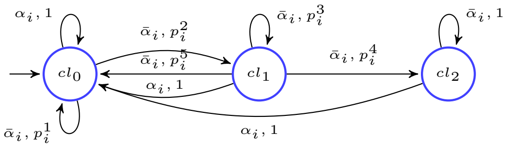
we let $s^{\tilde{M}_i} = cl_0$. The product MDP $\mathcal{M}_p^{\mathbf{N}_0} = \mathcal{M}^{\mathbf{N}_0} \otimes \mathcal{A}_\varphi$ then contains only 111 states. The initialization step takes 0.18 seconds, generating a control policy that ensures that $\mathcal{M}$ satisfies φ with probability 1. Hence, the synthesis process can be terminated with the total computation time of only 0.18 seconds.

Remark 2: In the case where there are multiple vehicles providing mobility on demand, the incremental approach will arbitrarily pick a policy for the vehicles whose full models have not been incorporated in the synthesis procedure. Suppose the synthesis procedure needs to be terminated due to the computational resource constraints. In this case, we can have these vehicles follow some preassigned policy.

Case Study 2: As another example, consider the problem illustrated in Figure 3 where an autonomous vehicle needs to go through a pedestrian crossing while there are 5 pedestrians who are already at or approaching the crossing. The vehicle is modeled by an MDP $\mathcal{M}_6$ whose state describes the cell occupied by the vehicle. For each $i \in \{0, \ldots, 4\}$, We define $L_6(c_i) = c_i^6$. The set *Act* captures the motion primitive of the vehicle and is defined as $Act = \{\alpha_0, \alpha_1\}$ where α_0 and α_1 correspond to the case where the vehicle applies the brake and the throttle, respectively. Pedestrian i is modeled by an MDP $\mathcal{M}_i$ whose state describes the cell occupied by the pedestrian and whose labeling function is defined as $L_i(c_j) = c_j^i$ for each $j \in \{0, \ldots, 4\}$. The models of the vehicle and the pedestrians are shown in Figure 3. The desired property of the system is that the vehicle never collides with any pedestrian and the vehicle eventually reaches cell c_4. This property can be expressed in LTL as $\varphi = \Box \bigwedge_{i \in \{1,\ldots,5\}, j \in \{0,\ldots,4\}} \neg(c_j^6 \wedge c_j^i) \wedge \Diamond c_4^6$. In this case, $\mathcal{A}_\varphi$ generated using ltl2dstar [21] contains 3 states.

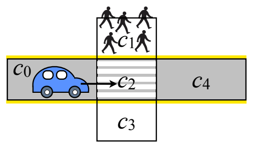


(a) Vehicle model $\mathcal{M}_4$.

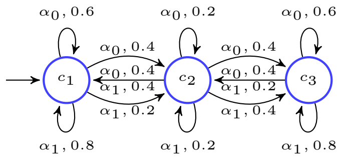


(b) Station models $\mathcal{M}_i, i \in \{1, 2, 3\}$. $p_i^1, \ldots, p_i^5 > 0$ are defined such that $\mathcal{M}_i$ is a valid MDP. $\bar{\alpha}_i$ can take any value in $Act \setminus \{\alpha_i\}$. State cl_2 is labeled as crowded.

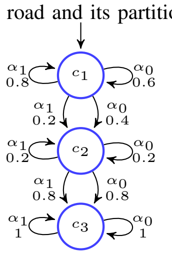
Fig. 2. The mobility-on-demand example.



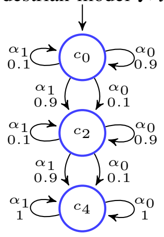
(a) The road and its partition



(b) Pedestrian model $\mathcal{M}_5$



(c) Pedestrian models $\mathcal{M}_1, \ldots, \mathcal{M}_4$



(d) Vehicle model $\mathcal{M}_6$

Fig. 3. The autonomous vehicle example.

The computation times required for control policy synthesis using the LP-based and SCC-based value iteration techniques described in Section IV are provided in Table II. Both approaches yield the probability of 0.54 that φ is satisfied under the synthesized control policy.

Let $\mathbf{N}_0 = \{6\}, \mathbf{N}_1 = \{1, 6\}, \mathbf{N}_2 = \{1, 2, 6\}, \ldots, \mathbf{N}_5 = \{1, \ldots, 6\}$, i.e., we successively add pedestrian $1, 2, \ldots, 5$ respectively, in each iteration. We consider 2 approaches: (1) the incremental approach described in Section V, and (2) the non-incremental approach where a control policy is recomputed from scratch once a pedestrian is added to the synthesis procedure in each iteration. For approach (2), we apply the LP-based technique since from Table II, it is the faster technique to solve this problem. For both approaches, 6 control policies $\mathcal{C}^{\mathbf{N}_0}, \ldots, \mathcal{C}^{\mathbf{N}_5}$ are generated for $\mathcal{M}^{\mathbf{N}_0}, \ldots, \mathcal{M}^{\mathbf{N}_5}$ respectively. The probabilities that the complete system $\mathcal{M}$ satisfies φ under these control policies are computed to be $\Pr_{\mathcal{M}}^{\mathcal{C}^{\mathbf{N}_0}}(\varphi) = 0.1$, $\Pr_{\mathcal{M}}^{\mathcal{C}^{\mathbf{N}_1}}(\varphi) = 0.12$, $\Pr_{\mathcal{M}}^{\mathcal{C}^{\mathbf{N}_2}}(\varphi) = 0.14$, $\Pr_{\mathcal{M}}^{\mathcal{C}^{\mathbf{N}_3}}(\varphi) = 0.15$, $\Pr_{\mathcal{M}}^{\mathcal{C}^{\mathbf{N}_4}}(\varphi) = 0.16$, and $\Pr_{\mathcal{M}}^{\mathcal{C}^{\mathbf{N}_5}}(\varphi) = 0.54$.

The comparison of various techniques discussed above is illustrated in Figure 4. For the incremental and non-incremental techniques, a jump in the probability occurs each time a new control policy is computed. Note that our incremental approach performs significantly better than any other techniques, generating an optimal solution within 21 seconds while the second fastest approach takes approximately 81 seconds to arrive at a solution with similar quality. This is largely due to the efficiency of our incremental construction of SCCs and AMECs. In addition to generating an optimal solution much more efficiently than other techniques, another key advantage of our incremental approach is its anytime nature. For large problems that cannot be fully solved, our incremental approach may still provide a reasonable, potentially sub-optimal solution.

Remark 3: In [16], [17], a similar autonomous vehicle problem as in Case Study 2 was considered. However, the pedestrians were modeled by Markov chains, preventing us from capturing the interaction between the pedestrians and the vehicle. Hence, the pedestrians behaved exactly the same

Technique	$\mathcal{M}_p$	SCCs	AMECs	Prob vector	Control policy	Total
LP	7.8	-	59.46	13.74	0.29	81.29
SCC-VI	7.8	55.04	59.46	0.91	0.29	123.5

TABLE II

AUTONOMOUS VEHICLE: TIME REQUIRED (IN SECONDS) FOR COMPUTING VARIOUS OBJECTS WHEN CONTROL POLICY SYNTHESIS IS SOLVED IN A SINGLE PASS.

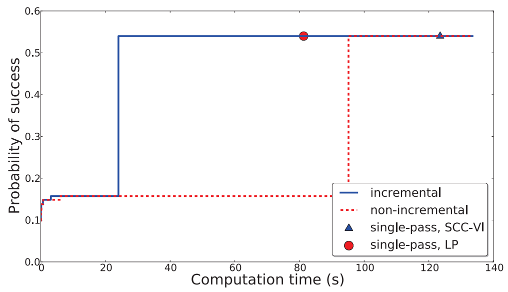


Fig. 4. Comparison of computation time and probability of satisfying the specification when control policies are synthesized using different techniques.

way no matter whether the vehicle applied the throttle or the brake, which is not reasonable. Modeling the pedestrians by MDPs allows us to capture such an interaction.

VII. CONCLUSIONS AND FUTURE WORK

An anytime algorithm for synthesizing a control policy for multi-agent systems with the objective of maximizing the probability of satisfying a given LTL specification is proposed. The agents are modeled as Markov decision processes and their interaction is captured by parallel composition. The proposed algorithm progressively computes a sequence of control policies, by taking into account only a small subset of agents initially and successively adding more agents to the synthesis procedure in each iteration until the constraint on computational resources is exceeded. Incremental construction of various objects needed to be computed during the synthesis procedure is also proposed to avoid unnecessary computation and exploit the objects computed in the previous iteration. The application of our approach is demonstrated in two case studies: mobility on demand and autonomy. In both case studies, our approach is able to obtain an optimal solution significantly faster than existing approaches. The anytime nature of our approach also provides an additional benefit. For large problems that cannot be fully solved, our incremental approach may still provide a reasonable, potentially sub-optimal solution.

Future work includes examining an effective approach to determine an agent to be added in each iteration. Such an agent may be picked based on the result from probabilistic verification. We are also considering integrating incremental synthesis with a complementary approach, namely distributed synthesis, where the synthesis problem is broken up into a set of smaller problems, each for each subsystem.

REFERENCES

[1] E. A. Emerson, "Temporal and modal logic," *Handbook of Theoretical Computer Science (Vol. B): Formal Models and Semantics*, pp. 995–1072, 1990.

[2] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems*. Springer-Verlag, 1992.

[3] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[4] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive robot control from abstraction and temporal logic specifications," *Special Issue of the IEEE Robotics & Automation Magazine on Formal Methods for Robotics and Automation*, vol. 18, pp. 65–74, 2011.

[5] T. Wongpiromsarn, S. Karaman, and E. Frazzoli, "Synthesis of provably correct controllers for autonomous vehicles in urban environments," in *IEEE Intelligent Transportation Systems Conference*, 2011.

[6] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *IEEE International Conference on Robotics and Automation*, pp. 3116–3121, 2007.

[7] G. Fainekos, H. Kress-Gazit, and G. Pappas, "Temporal logic motion planning for mobile robots," in *IEEE International Conference on Robotics and Automation*, pp. 2020–2025, 2005.

[8] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. Pappas, "Symbolic planning and control of robot motion [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 61–70, 2007.

[9] D. Conner, H. Kress-Gazit, H. Choset, A. Rizzi, and G. Pappas, "Valet parking without a valet," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007*, pp. 572–577, 2007.

[10] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc. of IEEE Conference on Decision and Control*, pp. 2222–2229, 2009.

[11] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2689–2696, 2010.

[12] X. C. Ding, S. L. Smith, C. Belta, and D. Rus, "LTL control in uncertain environments with probabilistic satisfaction guarantees," in *IFAC World Congress*, 2011.

[13] A. I. Medina Ayala, S. B. Andersson, and C. Belta, "Temporal logic control in dynamic environments with probabilistic satisfaction guarantees," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007*, pp. 3108–3113, 2011.

[14] Y. Chen, X. C. Ding, and C. Belta, "Synthesis of distributed control and communication schemes from global ltl specifications," in *Proc. of IEEE Conference on Decision and Control*, pp. 2718–2723, 2011.

[15] N. Ozay, U. Topcu, T. Wongpiromsarn, and R. M. Murray, "Distributed synthesis of control protocols for smart camera networks,," in *International Conference on Cyber-Physical Systems*, 2011.

[16] T. Wongpiromsarn, A. Ulusoy, C. Belta, E. Frazzoli, and D. Rus, "Incremental temporal logic synthesis of control policies for robots interacting with dynamic agents," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[17] A. Ulusoy, T. Wongpiromsarn, and C. Belta, "Incremental control synthesis in probabilistic environments with temporal logic constraints," in *Proc. of IEEE Conference on Decision and Control*, 2012.

[18] B. A. Brandin, R. Malik, and P. Malik, "Incremental verification and synthesis of discrete-event systems guided by counter examples," *IEEE Transactions on Automatic Control*, vol. 12, no. 3, pp. 387–401, 2004.

[19] R. C. Hill and D. M. Tilbury, "Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction," in *International Workshop on Discrete Event Systems*, pp. 399–406, 2006.

[20] K. T. Seow, "A dynamic programming approach to multi-level supervision," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 908–913, 2009.

[21] J. Klein and C. Baier, "Experiments with deterministic ω-automata for formulas of linear temporal logic," *Theoretical Computer Science*, vol. 363, pp. 182–195, 2006. Software available at `http://www.ltl2dstar.de/`.

[22] W. J. Mitchell, C. E. Borroni-Bird, and L. D. Burns., *Reinventing the Automobile: Personal Urban Mobility for the 21st Century*. Cambridge, MA: The MIT Press, 2010.

[23] F. Ciesinski, C. Baier, M. Größer, and J. Klein, "Reduction techniques for model checking markov decision processes," in *Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, pp. 45–54, 2008.

[24] M. Kwiatkowska, D. Parker, and H. Qu, "Incremental quantitative verification for markov decision processes," in *IEEE/IFIP International Conference on Dependable Systems & Networks*, pp. 359–370, 2011.