Formal Approach to the Deployment of Distributed Robotic Teams

Yushan Chen, *Student Member, IEEE*, Xu Chu (Dennis) Ding, *Member, IEEE*, Alin Stefanescu, *Member, IEEE*, and Calin Belta, *Member, IEEE*

***Abstract*—We present a computational framework for automatic synthesis of control and communication strategies for a robotic team from task specifications that are given as regular expressions about servicing requests in an environment. We assume that the location of the requests in the environment and the robot capacities and cooperation requirements to service the requests are known. Our approach is based on two main ideas. First, we extend recent results from formal synthesis of distributed systems to check for the distributability of the task specification and to generate local specifications, while accounting for the service and communication capabilities of the robots. Second, by using a technique that is inspired by linear temporal logic model checking, we generate individual control and communication strategies. We illustrate the method with experimental results in our robotic urban-like environment.**

***Index Terms*—Cooperative systems formal synthesis, robot control.**

I. Introduction

THE GOAL in robot motion planning and control is to be able to specify a motion task in a rich, high-level language and have the robot(s) automatically convert this specification into a set of low-level primitives, such as feedback controllers and communication protocols, to accomplish the task [2], [3]. In most existing works, the motion-planning problem is simply specified as "go from A to B, while avoiding obstacles" [3]. However, there are situations in which this is not enough to capture the nature of the task. Consider, e.g., the miniature robotic urban-like environment (RULE) shown in Fig. 1, where a robot might be required to "visit road R_1 or road R_2 without crossing intersection I_3 and then park in an available parking space," while at the same time obeying the traffic rules. Such a "rich" specification cannot be trivially converted to a sequence of "go from A to B" primitives.

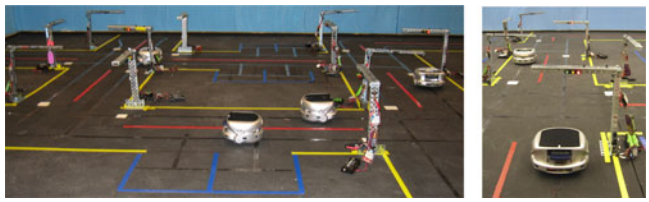


Fig. 1. Robotic urban-like environment. (Left) Khepera III car-like robots move autonomously on streets, while staying in their lanes, obeying traffic rules, and avoiding collisions. (Right) Car waiting at a traffic light.

When several robots are available, the problem becomes even more interesting and challenging. Assume that several service requests occur at different locations in the city, and they need to be serviced subject to some temporal and logical constraints. Some of these requests can be serviced by one (possibly specific) robot, while others require the collaboration of two or more (possibly specific) robots. For example, assume that the task is to first gather two pieces of data, one of which is available at P_3 only, and the other at either P_4 or P_5 and then fuse and transmit the data at one of the transmission locations P_1 or P_2. Assume that two robotic cars $\mathcal{A}_1$ and $\mathcal{A}_2$ are available; only $\mathcal{A}_1$ can read the data at P_4, and both cars are necessary to fuse and transmit the data. Can we generate provably correct individual control and communication strategies from such rich, global specifications? This is the problem that we address in this paper.

It has been advocated in [4]–[6] that temporal logics, such as linear temporal logic (LTL) and computation tree logic (CTL) [7], can be used as "rich" specification languages in mobile robotics. All of the aforementioned works suggest that the corresponding formal verification (model-checking) algorithms can be adapted for motion planning and controller synthesis from such specifications. A fundamental challenge in this area is to construct finite models that accurately capture the robot motion and control capabilities. Most current approaches are based on the notion of abstraction [8] and equivalence relations, such as simulation and bisimulation [9]. Enabled by recent developments in hierarchical abstractions of dynamical systems, it is now possible to model systems with linear dynamics [10], [11], polynomial dynamics [12], and nonholonomic (unicycle) dynamics [13] as finite transition systems (TS). Some related works show that such techniques can be extended to multiagent systems through the use of parallel composition [14]–[16]

Manuscript received November 10, 2010; revised April 8, 2011; accepted July 26, 2011. Date of publication September 12, 2011; date of current version February 9, 2012. This paper was recommended for publication by Associate Editor S. Carpin and Editor D. Fox upon evaluation of the reviewers' comments. This work was supported in part by the Office of Naval Research–Multidisciplinary University Research Initiative under N00014-09-1051, the Army Research Office under W911NF-09-1-0088, the Air Force Office of Scientific Research under YIP FA9550-09-1-020, and the National Science Foundation under CNS-0834260 at Boston University, Boston, MA, and by Romanian National Research Council-Executive Agency for Higher Education Research and Innovation Funding 7/05.08.2010 at the University of Pitesti, Pitesti, Romania. Preliminary results from this paper were presented at the 10th International Symposium on Distributed Autonomous Robotics Systems.

Y. Chen is with the Department of Electrical Engineering, Boston University, Boston, MA 02215 USA (e-mail: yushanc@bu.edu).

X. C. Ding and C. Belta are with the Department of Mechanical Engineering, Boston University, Boston, MA 02215 USA (e-mail: xcding@bu.edu; cbelta@bu.edu).

A. Stefanescu is with Department of Computer Science, University of Pitesti, Pitesti 110040, Romania (e-mail: alin@stefanescu.eu).

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.2011.2163434

or reactive games [17]. However, such bottom-up approaches are expensive and can lead to state-space explosion, even for relatively simple problems.

As a result, one of the main challenges in the area of motion planning and control of distributed teams that are based on formal methods is to create provably correct, top-down approaches in which a global, "rich" specification can be decomposed into local (individual) specifications, which can then be used to automatically synthesize robot control and communication strategies. In this paper, we draw inspiration from the area of distributed formal synthesis [18] to develop such a top-down approach. We consider a team of robots that can move among the regions of a partitioned environment and have known capabilities of servicing a set of requests that can occur in the regions of the partition. Some of these requests can be serviced by a robot individually, while some require the cooperation of groups of robots. We present a framework that allows for the fully automatic synthesis of robot control and communication strategies from a task specification that is given as a regular expression (RE) over the set of requests. The problem that we consider is purely discrete, where the (partitioned) environment is modeled as a discrete graph and the robots as agents that can move between adjacent vertices. Our solution is quite general and can be used in conjunction with abstraction techniques to control and deploy multiple agents with continuous dynamics.

The contribution of this study is threefold. First, we develop a top-down computational framework for automatic deployment of mobile agents from global specifications that are given as REs over environmental requests. This is a significant improvement of our recent work [19] by enlarging the class of specifications for which a solution exists. Specifically, we show how a satisfying distributed execution can be found when the global specification is a traced-closed language, rather than the more restrictive product language as in [19]. Second, we provide a relaxation to the standard approach of distributed synthesis modulo synchronous products (SPs) and language equivalence [18]. To this end, this paper extends upon our previous work [20], in which we provided two heuristics for the case of asynchronous automata. Third, we implement and illustrate the computational framework in our Khepera-based RULE (see Fig. 1). In this experimental setup, the robots can be automatically deployed from specifications that are given as REs over requests occurring at regions in a miniature city.

Our framework is significantly less expensive than the bottom-up approaches [14]–[16] in terms of computational complexity, since the construction of the parallel composition of the individual motions is not necessary, and the state-space explosion problem is avoided. Arguably, the closest related work is [21], where the global specifications that are given as languages over a set of events were checked for distributability modulo bisimulation, which is more restrictive than distributability modulo language equivalence. Moreover, the expressivity of the specifications in [21] was restricted to a subset of regular languages (i.e., languages that are accepted by TS). In addition, the robot motion capabilities and possible deadlocks that are caused by parallel executions of the robots were not taken into consideration.

Some of the results in this paper were presented without proofs in [1]. In this paper, we include all technical details that are omitted, relax some of the assumptions in [1], and include a complexity analysis of the overall approach. The remainder of the paper is organized as follows. Some preliminaries are introduced in Section II. The problem is formulated in Section III. An outline of our approach is described in Section IV. An algorithm for the distribution of the task specification over a robotic team and synthesis of individual control and communication strategies is presented in Section V. In Section VI, we discuss the computational complexity of our approach. In Section VII, we show that some of the assumptions that we made to keep the notation and computational complexity to a minimum can be relaxed to accommodate more realistic scenarios. Experimental case studies are presented in Section VIII. We conclude with final remarks and directions for future work in Section IX.

II. Preliminaries

For a set Σ, we use $|\Sigma|$ and 2^{Σ} to denote its cardinality and power set, respectively. A word, i.e., $w = w(0)w(1)\cdots w(n)$, over a set Σ is a sequence of symbols from Σ. We use Σ^* to denote the set of all finite words over Σ. The length of a word $w \in \Sigma^*$ is denoted by $|w|$. A language is a set of words.

Definition 2.1 (Transition System): A TS is a tuple, i.e., $T = (S, s_0, \rightarrow, \Pi, \models)$, where S is the finite set of states, $s_0 \in S$ is the initial state, $\rightarrow \subseteq S \times S$ is the transition relation, Π is the finite set of atomic propositions (observations), and $\models \subseteq S \times \Pi$ is the satisfaction relation.

A transition $(s, s') \in \rightarrow$ is also denoted by $s \rightarrow s'$. For an arbitrary state $s \in S$, we define $\Pi_s = \{\pi \in \Pi \mid (s, \pi) \in \models\} \in 2^{\Pi}$ as the set of all atomic propositions that are satisfied at s. A trajectory of T is a sequence $s(0)s(1)\cdots s(n)$ with the property that $s(0) = s_0$, $s(i) \in S$, and $s(i) \rightarrow s(i+1)$, for all $i = 0, \ldots, n-1$. We say that a trajectory, i.e., $s = s(0)s(1)\cdots s(n)$, of T satisfies a word, i.e., $w = w(0)w(1)\cdots w(n)$, if $w(i) \in \Pi_{s(i)}$, for all $i = 0, \ldots, n$.

Definition 2.2 (Finite-State Automaton): A finite-state automaton (FSA) is a tuple, i.e, $A = (Q, q_0, \Sigma, \rightarrow_A, F)$, where Q is the set of states, $q_0 \in Q$ is the initial state, Σ is the set (alphabet) of actions, $\rightarrow_A \in Q \times \Sigma \times Q$ is the transition relation, and $F \subseteq Q$ is the set of final (accepting, marked) states. An FSA A is a weighted FSA if there is a nonnegative-valued weight function that is defined on the transitions of A.

We also write $q \xrightarrow{\sigma}_A q'$ to denote $(q, \sigma, q') \in \rightarrow_A$. A *run* of an FSA on a finite word, i.e., $w = \sigma_0\sigma_1 \ldots \sigma_m \in \Sigma^*$, is a sequence of states $q_0q_1 \ldots q_{m+1}$, such that $q_i \xrightarrow{\sigma_i} q_{i+1}$, for all $i = 0, 1, \ldots, m$. A finite word w is *accepted* by an FSA if there exists a run on it: $q_0q_1 \ldots q_{m+1}$ satisfying $q_{m+1} \in F$. The *language* accepted by an FSA A (the language of A), which is denoted by $\mathcal{L}(A)$, is the set of all finite words that are accepted by A. Two FSAs over the same set of actions are called *language equivalent* if they accept the same language.

A *deterministic* FSA (DFA) is an FSA, where for each $q \in Q$ and $\sigma \in \Sigma$, there exists at most one $q' \in Q$, such that $q \xrightarrow{\sigma}_A q'$. Otherwise, the FSA is called *nondeterministic* FSA (NFA). Another extension of the FSA, which is denoted as ϵ-NFA, is an

NFA with ϵ, i.e., the empty string, as a possible input. Note that given any ϵ-NFA, one can construct an equivalent DFA that accepts the same language as the ϵ-NFA. The construction of a DFA, given a ϵ-NFA, is based on a well-known subset construction algorithm, which incorporates ϵ-transitions through the mechanism of ϵ-closure. Furthermore, given a DFA, there exists a unique (up to isomorphism) *minimal* DFA that is language equivalent with the initial DFA, and well-known, efficient minimization algorithms are available. See [22] for more details about the algorithm of construction of a *minimal* DFA given a ϵ-NFA.

The language that is accepted by an FSA is called a *regular language*. An RE is a concise representation of a regular language. We use L_ϕ to denote the language that satisfies an RE ϕ. Informally, an RE over a set Σ is defined recursively by using three standard operators: union (denoted by $+$), concatenation, and iteration (denoted by *). For example, with $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$, the RE $(\sigma_1 + \sigma_2 + \sigma_3)^*\sigma_1(\sigma_1 + \sigma_2 + \sigma_3)^*\sigma_1(\sigma_1 + \sigma_2 + \sigma_3)^*$ specifies that action σ_1 should be executed at least twice, while the RE $(\sigma_1 + \sigma_2)^*\sigma_1(\sigma_1 + \sigma_2)^*$ requires that action σ_1 should be executed at least once and that action σ_3 is forbidden. Finally, $\sigma_1\sigma_2 + \sigma_2\sigma_1$ specifies that actions σ_1 and σ_2 need to be executed exactly once in an arbitrary order.

Given an RE, a DFA that accepts all and only the words that satisfy the RE can be constructed by using an off-the-shelf tool, such as Java Formal Languages & Automata Package (JFLAP) [23]. Given a regular language $\mathcal{L}(A)$ over Σ, which is accepted by a DFA A, the complement of $\mathcal{L}(A)$ is defined as $\overline{\mathcal{L}(A)} := \Sigma^*\backslash\mathcal{L}(A)$. Note that a DFA $\neg A$, which is defined as a DFA that accepts the language $\overline{\mathcal{L}(A)}$, can be constructed by swapping the accepting states of A with its nonaccepting states.

Definition 2.3 (Distribution): Given a set Σ, a collection of subsets, i.e., $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$, where I is an index set, is called a distribution of Σ if $\cup_{i\in I}\Sigma_i = \Sigma$. For $\sigma \in \Sigma$, we denote $I_\sigma = \{i \in I \mid \sigma \in \Sigma_i\}$.

For a word $w \in \Sigma^*$ and a subset $S \subseteq \Sigma$, let $w{\restriction}_S$ denote the *projection* of w onto S, which is obtained by erasing all actions σ in w that do not belong to S. For a language $L \subseteq \Sigma^*$ and a subset $S \subseteq \Sigma$, let $L{\restriction}_S$ denote the projection of L onto S, which is given by $L{\restriction}_S := \{w{\restriction}_S \mid w \in L\}$. Starting from the observation that the projection of a regular language is a regular language, the projection of an FSA A on a subset $S \subseteq \Sigma$ is another FSA (which is denoted by $A{\restriction}_S$) that accepts the language $\mathcal{L}(A){\restriction}_S$. The projection of an FSA can be constructed through the process of ϵ-closure, determinization, and minimization (see [24]).

Definition 2.4 (Product Language): Given a distribution, i.e., $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$, of Σ, the product of a set of languages L_i over Σ_i is denoted by $\|_{i\in I}\, L_i$ and defined as $\|_{i\in I}\, L_i := \{w \in \Sigma^* \mid w{\restriction}_{\Sigma_i} \in L_i \text{ for all } i \in I\}$. A *product language* over a distribution Δ of Σ is a language L such that $L = \|_{i\in I}\, L_i$, where $L_i = L{\restriction}_{\Sigma_i}$ for all $i \in I$.

Definition 2.5 (Trace-Closed Language): Given a distribution, i.e., $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$, of Σ and $w, w' \in \Sigma^*$, we say that w is trace equivalent to w' ($w \sim_\Delta w'$) iff $w{\restriction}_{\Sigma_i} = w'{\restriction}_{\Sigma_i}, \forall i \in I$. Let $[w]_\Delta$ denote the trace-equivalence class of $w \in \Sigma^*$. A *trace-closed language* over a distribution Δ of Σ is a language L such that for all $w \in L$, $[w]_\Delta \subseteq L$. For an arbitrary language L, we denote $L_\Delta := \{w \in L | [w]_\Delta \subseteq L\}$ as the largest trace-closed subset of L.

The class of trace-closed languages is closed under the operations of union, intersection, and complementation. Obviously, if L is trace closed with respect to a distribution Δ, then $L = L_\Delta$. Note that a product language is trace closed but the converse is not true. See [24]–[26] for more details on trace-closed and product languages.

III. Problem Formulation

Let

$$\mathcal{E} = (V, \rightarrow_\mathcal{E}) \tag{1}$$

be an environment graph, where V is the set of vertices, and $\rightarrow_\mathcal{E} \subseteq V \times V$ is a relation that models the set of edges, e.g., $\mathcal{E}$ can be the quotient graph of a partitioned environment, where V is a set of labels for the regions in the partition, and $\rightarrow_\mathcal{E}$ is the corresponding adjacency relation. Assume that we have a team of robots (moving agents) $\mathcal{A}_i$, $i \in I$, whose motions are restricted by $\mathcal{E}$, where I is a set of robot labels. Let Σ be a set of service requests, or actions that are to be performed at the vertices of $\mathcal{E}$. To keep notation to a minimum, we assume for now that the locations of the service requests are defined as a function $a : \Sigma \rightarrow V$ (i.e., different requests can occur at the same vertex but vertices do not share requests; there may be no request at some vertices of $\mathcal{E}$). Later in this paper (see Section VII), we discuss how this assumption can be relaxed.

We model the capacity of the robots to service requests and the cooperation requirements among the robots as a distribution, i.e., $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$ of Σ (see Definition 2.3). Σ_i represents the set of requests that can be serviced by the robot $\mathcal{A}_i$. For a given request $\sigma \in \Sigma$, $I_\sigma = \{i \in I \mid \sigma \in \Sigma_i\}$ is the set of labels of all the agents that can service it. The semantics of this distribution is defined as follows. For an arbitrary request σ, if $|I_\sigma| = 1$ (i.e., there is only one agent that owns it), the agent can (and should) service the request by itself, independent of the other agents. This kind of request is called an *independent* request. If $|I_\sigma| > 1$, all the agents $\mathcal{A}_i$ with $i \in I_\sigma$ must service the request simultaneously (i.e., they need to communicate to service σ together). This kind of request is called a *shared* request. An agent is said to service a request σ if it visits the vertex $a(\sigma)$. For the simplicity of presentation, we assume for now that two or more robots sharing a request σ can communicate at the (only) vertex $a(\sigma)$, where σ occurs. In Section VII, we discuss how we can accommodate arbitrary communication graphs.

Remark 3.1: The distribution uniquely defines the cooperation requirements among the robots, e.g., if a request is in both Σ_1 and Σ_2, it requires the cooperation between robots $\mathcal{A}_1$ and $\mathcal{A}_2$. Imagine a scenario, where multiple robots are able to service a request that only requires one robot. In this case, the distribution that describes the capability of robots servicing the requests is not unique. In this paper, we only consider a fixed given distribution. We will address the removal of this limitation in future work.

We model the motion capabilities of each agent $\mathcal{A}_i$, $i \in I$, on the environment graph $\mathcal{E}$ using a TS T_i (see Definition 2.1),

which is defined as follows:

$$T_i = (V, v_{0_i}, \rightarrow_i, \Pi, \models_i), \quad i \in I \tag{2}$$

where $v_{0_i} \in V$ is the initial position of $\mathcal{A}_i$; $\rightarrow_i$ is a reflexive transition relation that satisfies $\rightarrow_i \subseteq \rightarrow_{\mathcal{E}} \cup_{v \in V} \{(v, v)\}$; $\Pi = \Sigma \cup \{\epsilon\}$ (ϵ is the empty request); and $\models_i \subseteq V \times \Pi$ is a relation, where $(v, \epsilon) \in \models_i$ for all $v \in V$ and $(v, \sigma) \in \models_i$, $\sigma \in \Sigma_i$, if and only if $v = a(\sigma)$. In other words, the motion of robot $\mathcal{A}_i$ is restricted by the transition relation $\rightarrow_i$, which captures motion (actuation) constraints in addition to $\rightarrow_{\mathcal{E}}$. The locations of the requests in the environment are captured by the relation $\models_i$. As will become clear later, each vertex that satisfies ϵ captures that a robot can pass through a vertex without servicing any request.

Definition 3.1 (Motion and Service Plan): A motion and service (MS) plan for robot $\mathcal{A}_i$, $i \in I$ is a word $ms_i \in (V \cup \Sigma_i)^*$ that satisfies the following conditions.

1) $ms_i(0) = v_{0_i}$.
2) If $ms_i(j) \in \Sigma_i$, then $ms_i(j-1) \in V$, and $(ms_i(j-1), ms_i(j)) \in \models_i$, for all $j > 1$.
3) $ms_i{\restriction_V}$ is a trajectory of T_i.

An MS plan for robot $\mathcal{A}_i$ uniquely defines a *motion plan* $m_i = ms_i{\restriction_V}$ and a *service plan* $s_i = ms_i{\restriction_{\Sigma_i}}$. We say that a service plan s_i can be *implemented* by robot $\mathcal{A}_i$ if there exists an MS plan ms_i such that $ms_i{\restriction_{\Sigma_i}} = s_i$. The semantics of an MS plan is as follows. A vertex entry $ms_i(j) \in V$ means that the vertex $ms_i(j)$ should be visited. A request entry $ms_i(j) \in \Sigma_i$ means that robot $\mathcal{A}_i$ should service the request $ms_i(j)$ at the vertex $ms_i(j-1)$. A shared request $ms_i(j)$ (i.e., $|I_{ms_i(j)}| > 1$) triggers a *wait-and-leave* protocol: At vertex $ms_i(j-1)$, robot $\mathcal{A}_i$ broadcasts the request $ms_i(j)$ and listens for the broadcasts of $ms_i(j)$ from all other agents $\mathcal{A}_j$, $j \in I_{ms_i(j)} \setminus \{i\}$. When they all are received, the request $ms_i(j)$ is serviced, and then, $\mathcal{A}_i$ moves to the next vertex.

Remark 3.2: We assume that interrobot communication is always possible. Note that one robot only needs to synchronize (using the wait-and-leave protocol introduced earlier) with other robots that share a request σ, before servicing this shared request. The loose synchronization enables parallel executions of individual agents.

Given a set of MS plans $\{ms_i, i \in I\}$ for the robot team, there may exist many possible sequences of requests that are serviced by the team because of parallel executions. (We do not assume that we know the time it takes for each agent to service requests.)

Definition 3.2 (Global Behavior of the Team): Given a set of MS plans $\{ms_i, i \in I\}$, we denote

$$L_{\mathrm{MS}}^{\mathrm{team}}(\{ms_i, i \in I\}) := \|_{i \in I} \{s_i\} \tag{3}$$

as the set of all possible sequences of requests that are serviced by the team of robots, while they follow their individual MS plans.

For simplicity of notation, we use $L_{\mathrm{MS}}^{\mathrm{team}}$ for $L_{\mathrm{MS}}^{\mathrm{team}}(\{ms_i, i \in I\})$ when there is no ambiguity.

Definition 3.3 (Satisfying set of MS Plans): A set of MS plans $\{ms_i, i \in I\}$ satisfies a specification given as an RE ϕ over Σ if and only if $L_{\mathrm{MS}}^{\mathrm{team}} \neq \emptyset$ and $L_{\mathrm{MS}}^{\mathrm{team}} \subseteq L_\phi$.

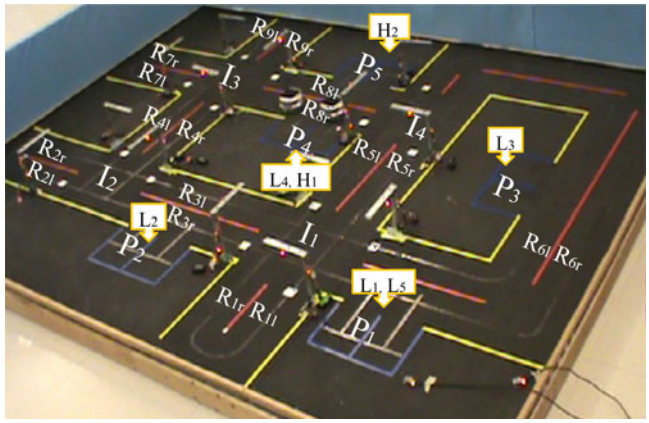


Fig. 2. City for the case study. The topology of the city, the requests that occur at the parking lots and the road, intersection, and parking lot labels.

Remark 3.3: For a set of MS plans, the corresponding $L_{\mathrm{MS}}^{\mathrm{team}}$ could be an empty set by the definition of product of languages (since there may not exist a word $w \in \Sigma^*$, such that $w{\restriction_{\Sigma_i}} = s_i \ \forall i \in I$). In practice, this case corresponds to a scenario, where one (or more) agent waits indefinitely for other agents to service a request σ that is shared among these agents. For example, if σ does not appear in the service plan of one of the agents who own σ, but it appears in the service plans of some other agents, then all those agents will be stuck in a "deadlock" state and wait indefinitely. As another example, let $s_1 = \sigma_1\sigma_2$, $s_2 = \sigma_2\sigma_1$, and $\Sigma_1 = \Sigma_2 = \{\sigma_1, \sigma_2\}$. In this case, robots $\mathcal{A}_1$ and $\mathcal{A}_2$ will wait for each other indefinitely. When a deadlock occurs, the set of MS plans is not satisfying.

We are now ready to formulate the main problem.

Problem 3.1: Given a team of agents $\mathcal{A}_i$, $i \in I$ with motion capabilities T_i [see (2)] on a graph $\mathcal{E}$ [see (1)], a set of service requests Σ, a function $a : \Sigma \rightarrow V$ that shows the location of the service requests, a distribution, i.e., $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$, of Σ that models the capacity of the robots to service requests and the cooperation requirements among the robots, and a task specification ϕ in the form of an RE over Σ, find a satisfying set of MS plans $\{ms_i, i \in I\}$.

Case Study 1: For illustration, throughout this paper, we consider an example in our RULE (see Fig. 2). Modeling RULE by the use of the proposed framework proceeds as follows. The set of vertices V of the environment graph $\mathcal{E}$ is the set of labels that are assigned to the roads, intersections, and parking lots. The edges in $\rightarrow_{\mathcal{E}}$ show how these regions are connected. We consider two robots (Khepera III miniature cars) running in the environment, whose motion capabilities can be modeled as a TS T_i, which is shown in Fig. 3, where $\rightarrow_i = \rightarrow_{\mathcal{E}}$ captures how the robot can move among adjacent regions. Note that these transitions are, in reality, enabled by low-level control primitives (see Section VIII). We assume that the selection of a control primitive at a region uniquely determines the next region. This corresponds to a deterministic (control) TS, in which each trajectory of T_i can be implemented by the robot in the environment by using the sequence of corresponding motion primitives.

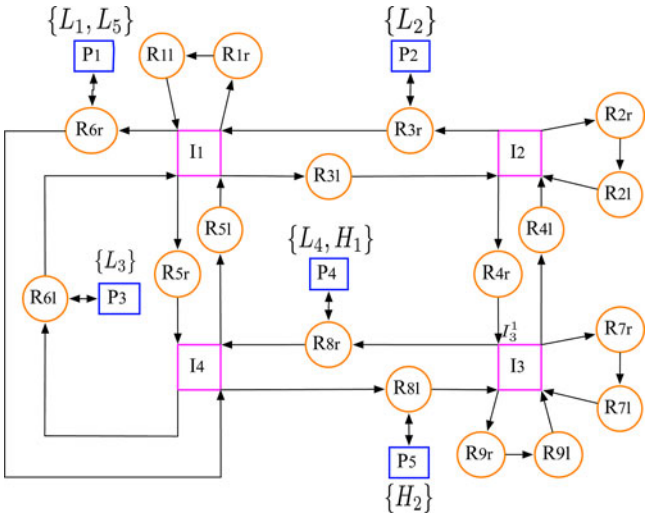


Fig. 3. Transition systems T_i that capture the motion capabilities of the robots, which are identical, except for the initial state (not shown).

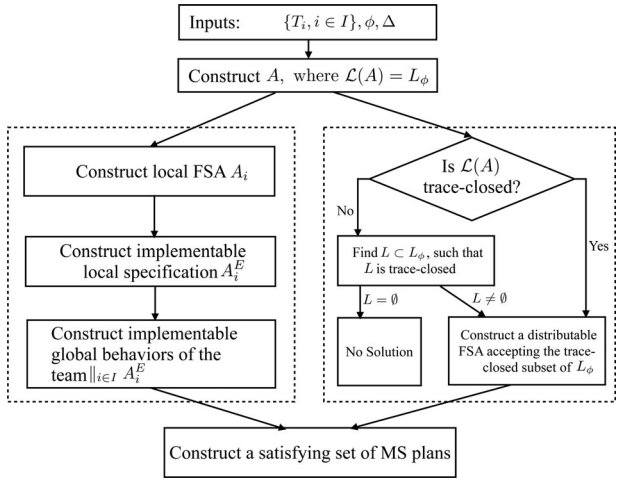


Fig. 4. Schematic representation of our approach to Problem 3.1.

Assume that the set of service requests is given as $\Sigma = \{H_1, H_2, L_1, L_2, L_3\}$, where L_i's represent pieces of data that can be collected in parallel by a single robot, while H_i's represent data fusion and decision making processes, which require the cooperation of the two robots. The distribution, i.e., $\Sigma_1 = \{L_1, H_1, H_2\}, \Sigma_2 = \{L_2, L_3, H_1, H_2\}$, captures the robots' capabilities to collect the data and cooperation requirements for the data fusion. Assume that the requests occur at the parking lots as shown in Fig. 2. The relation $\models_i$ indicates the locations of the requests. We want to accomplish the following task: "Fuse the initial information carried by two robots (H_1); collect data at P_1 (L_1) and P_2 (L_2) in an arbitrary order; fuse the collected data at P_5 (H_2); and finally, collect data from P_1 (L_1) and P_3 (L_3) in an arbitrary order." Such a task translates to the following RE:

$$\phi: \; H_1 \; (L_1 L_2 \; + \; L_2 L_1) \; H_2 \; (L_1 L_3 \; + \; L_3 L_1). \qquad (4)$$

IV. Outline of the Approach

Our approach to solve Problem 3.1 can be summarized as follows. We first generate "implementable" global behaviors of the team, which capture all the service plans that can be implemented by the robots (see Section V-A). Then, if the language L_ϕ that satisfies the global specification ϕ is trace closed, we generate a solution to the problem. Otherwise, we attempt to find a subset of L_ϕ that is trace closed. If we succeed (i.e., the obtained subset is not empty), then we use it to generate a solution (see Section V-B). We illustrate our approach in Fig. 4.

In our previous work [19], we provided a solution to Problem 3.1 through an extension to regular languages of the standard approach to distributed synthesis modulo SPs and language equivalence for TS [18]. As stated in [18], if the language that satisfies ϕ is a product language, then we can construct a set of local specifications, such that when they synchronize, they are equivalent to the global specification. At first look, one might think that we can generate control and communication strategies individually for robot $\mathcal{A}_i$ given such a local specification and the motion capabilities of $\mathcal{A}_i$. However, such a purely top-down approach will not work because of the "deadlock" scenario that is described in Remark 3.3, i.e., we cannot guarantee that the motion of the team $L_{\mathrm{MS}}^{\mathrm{team}} \neq \emptyset$. In [19], we approached the "deadlock" problem through an additional (computationally expensive) synchronization process. However, our approach was conservative since we could only generate a solution to Problem 3.1 for the case when the language that satisfies ϕ was a product language over the given distribution (see Definition 2.4).

In this paper, we propose a solution to Problem 3.1, which is complete if the language that satisfies ϕ is trace closed over the given distribution. Since trace-closed languages are less restrictive than product languages (i.e., product languages are trace closed but not vice versa), we reduce the conservativeness of our previous approach. Furthermore, our proposed solution is less computationally expensive. Indeed, to check whether a language is trace closed is linear in the size of the FSA accepting the language, while to check whether a language is a product language is polynomial space (PSPACE) complete [24]. Last but not least, in this paper, we attempt to find a solution even when the language that satisfies ϕ is not trace closed over the given distribution (in which case, our previous approach cannot provide a solution).

V. Synthesis of Local Motion and Service Plans From the Global Specification

A. Synthesis of Implementable Global Behaviors

We begin with the conversion of the specification ϕ over Σ to a minimal DFA, i.e., $A = (Q, q_0, \Sigma, \rightarrow, F)$, which accepts exactly the language over Σ that satisfies ϕ (using JFLAP [23]). We call A the *global* specification. Given the distribution Δ, we assign requests to each agent. Specifically, we construct a set of projected FSAs, i.e., $A_i = (Q_i, q_{0_i}, \Sigma_i, \rightarrow_{A_i}, F_i)$, whose languages are the projections of $\mathcal{L}(A)$ onto the local alphabets Σ_i, $i \in I$. (See Section II for the construction of A_i.) The projected FSAs are used as a starting point to find a solution to Problem 3.1 because of the following proposition.

Proposition 5.1: If a set of MS plans $\{ms_i, i \in I\}$ is a solution to Problem 3.1, then its corresponding service plans, i.e., $s_i = ms_i\restriction_{\Sigma_i}$, are accepted words of A_i for all $i \in I$.

Proof: If $\{ms_i, i \in I\}$ is a solution to Problem 3.1, then we have $\|_{i\in I} s_i \subseteq \mathcal{L}(A)$ and $\|_{i\in I} s_i \neq \emptyset$. We can find a word $w \in \|_{i\in I} s_i \subseteq \mathcal{L}(A)$, such that $[w]_\Delta = \|_{i\in I} s_i$, where $s_i = w\restriction_{\Sigma_i}$ for all $i \in I$. By the definition of the projection of A onto a distribution, $w\restriction_{\Sigma_i} \in \mathcal{L}(A_i)$, and thus, $s_i \in \mathcal{L}(A_i)$.

However, to provide a provably correct solution for Problem 3.1, it is not sufficient to simply choose an arbitrary accepted word from the projected FSAs A_i to be a service plan s_i. We need to make sure that 1) the service plan s_i can be implemented by robot $\mathcal{A}_i$, and 2) all possible sequences of requests that are serviced by the team satisfy ϕ. To satisfy the first requirement, we aim to model the implementable global behaviors of the team. To achieve this, we first obtain an "implementable local" specification A_i^E for each $i \in I$, such that the language of A_i^E equals the set of all the accepted words of A_i that can be implemented by the agent $\mathcal{A}_i$. We address the second requirement in Section V-B.

To obtain A_i^E, we construct a new FSA $\widehat{A}_i$ from $A_i = (Q_i, q_{0_i}, \Sigma_i, \rightarrow_{A_i}, F_i)$ by adding action ϵ to Σ_i and self-transitions (q, ϵ, q) to each state $q \in Q_i$. For a robot, ϵ means that no request is serviced. We denote the set of all these self-transitions by $\rightarrow_{\epsilon_i}$. The FSA $\widehat{A}_i$ can now be defined as

$$\widehat{A}_i = (\widehat{Q}_i, \widehat{q}_{0_i}, \widehat{\Sigma}_i, \rightarrow_{\widehat{A}_i}, \widehat{F}_i) \tag{5}$$

where $\widehat{Q}_i = Q_i, \widehat{q}_{0_i} = q_{0_i}, \widehat{\Sigma}_i = \Sigma_i \cup \{\epsilon\}, \rightarrow_{\widehat{A}_i} = \rightarrow_{A_i} \cup \rightarrow_{\epsilon_i}$, and $\widehat{F}_i = F_i$. It is important to note that these self-transitions do not affect the semantics of A_i, since they mean that if no request is served by robot $\mathcal{A}_i$, then the state of A_i remains the same. Given a word $\widehat{w}$ that is accepted by $\widehat{A}_i$, we can obtain a word, i.e., $w = \widehat{w}\restriction_{\Sigma_i}$, accepted by A_i by treating ϵ as an empty string. Note that input ϵ corresponds to the observation ϵ in the TS T_i, and the set of inputs $\widehat{\Sigma}_i$ of $\widehat{A}_i$ is a subset of the observations Π of T_i.

To restrict the trajectories of a TS T_i with a set of observations Π to the language that is accepted by an FSA with a set of actions $\widehat{\Sigma}_i \subseteq \Pi$, we define the following product automaton, which is inspired by LTL model checking [7]:

Definition 5.1 (Adapted from [4]): The product automaton, i.e., $P_i = T_i \otimes \widehat{A}_i$, between a TS $T_i = (V, v_{0_i}, \rightarrow_i, \Pi, \models_i)$ and an FSA $\widehat{A}_i = (\widehat{Q}_i, \widehat{q}_{0_i}, \widehat{\Sigma}_i, \rightarrow_{\widehat{A}_i}, \widehat{F}_i)$, where $\widehat{\Sigma}_i \subseteq \Pi$, is an FSA $P_i = (Q_{P_i}, q_{0_{P_i}}, \Sigma_{P_i}, \rightarrow_{P_i}, F_{P_i})$, where $Q_{P_i} = V \times \widehat{Q}_i$, $q_{0_{P_i}} = (v_{0_i}, \widehat{q}_{0_i})$ is the initial state, $\Sigma_{P_i} = \widehat{\Sigma}_i$ is the set of inputs, and $F_{P_i} = V \times \widehat{F}_i$ is the set of accepting (final) states. The transition relation $\rightarrow_{P_i} \subseteq Q_{P_i} \times \Sigma_{P_i} \times Q_{P_i}$ is defined as $(v, q) \xrightarrow{\sigma_{P_i}}_{P_i} (v', q')$ if and only if $v \rightarrow_i v', q \xrightarrow{\sigma_{P_i}}_{\widehat{A}_i} q'$, and $\sigma_{P_i} \in \Pi_v$.

A transition $(v, q) \xrightarrow{\sigma}_{P_i} (v', q')$ of P_i exists if and only if $(v, v') \in \rightarrow_i$ and request σ occurs at the vertex v. Transitions with input ϵ mean that a robot is moving from a vertex v to a vertex v' (v may be equal to v') without servicing any request. $r_{P_i} = (v_i(0), \widehat{q}_i(0))(v_i(1), \widehat{q}_i(1)) \cdots (v_i(n), \widehat{q}_i(n))$, where $\widehat{q}_i(j) \in \widehat{Q}_i$, $v_i(j) \in V$, and $j \in \{1, \ldots, n\}$ is a run accepted by the product automaton $P_i, i \in I$. An accepted run r_{P_i} can be easily found using a backward reachability search that starts from all states in F_{P_i} and ends at the initial state $q_{0_{P_i}}$. We define the projection of r_{P_i} onto T_i as $\gamma_{T_i}(r_{P_i}) = v_i(0)v_i(1) \cdots v_i(n)$. The following proposition shows that we can use a run of P_i to find a trajectory of T_i that satisfies the local specification (a word of $\widehat{A}_i$).

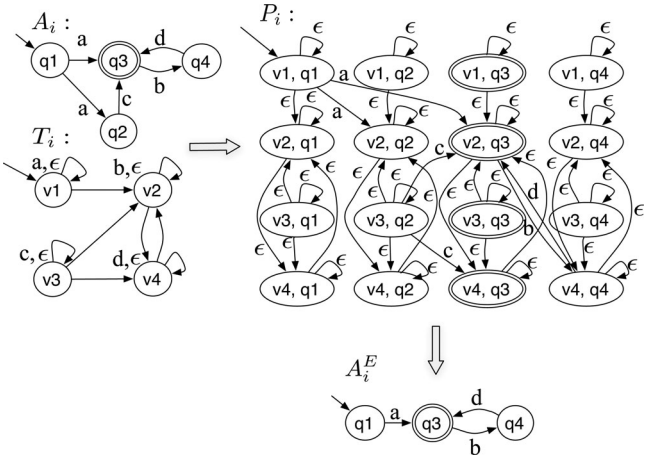


Fig. 5. Example of construction of A_i^E from T_i and A_i. We first generate $\widehat{A}_i$ from A_i, and then, we obtain P_i as defined in Definition 5.1. A_i^E is P_i after ϵ-closure, determinization, and minimization. For example, word $ac \in \mathcal{L}(A_i)$ cannot be implemented by T_i, and thus, it is not accepted by A_i^E.

Proposition 5.2: Given any word $w_{\widehat{A}_i} \in \mathcal{L}(\widehat{A}_i)$, there exists at least one trajectory of T_i that satisfies $w_{\widehat{A}_i}$ if and only if $w_{\widehat{A}_i} \in \mathcal{L}(P_i)$.

Proof "$\Longleftarrow$": Given a word $w_{\widehat{A}_i} \in \mathcal{L}(P_i)$; then, there exists a run r_{P_i} of P_i that generates $w_{\widehat{A}_i}$. The projection $\gamma_{T_i}(r_{P_i})$ is a trajectory r_{T_i} of T_i that satisfies the language of $\widehat{A}_i$ (by definition of the product automaton). Hence, there exists a trajectory of T_i, which satisfies $w_{\widehat{A}_i}$.

"$\Longrightarrow$": Given a word $w_{\widehat{A}_i} = w(0)w(1) \cdots w(n)$ accepted by $\widehat{A}_i$ and a trajectory $r_{T_i} = v(0)v(1) \cdots v(n)$ of T_i that satisfies $w_{\widehat{A}_i}$; then, we have $v(j) \rightarrow_i v(j+1)$ and $w(j) \in \Pi_{v(j)}$ for all $j \in \{0, \ldots, n-1\}$. Since the transition relation $\rightarrow_i$ of T_i is a reflexive transition relation, there is always a transition stating at every state. Hence, for $v(n)$, we can always find a vertex $v(n+1)$, such that $v(n) \rightarrow_i v(n+1)$. Therefore, given $w_{\widehat{A}_i}$, we can find an accepted run $r_{\widehat{A}_i} = \widehat{q}(0)\widehat{q}(1) \cdots \widehat{q}(n+1)$ of $\widehat{A}_i$, which generates $w_{\widehat{A}_i}$. According to Definition 5.1, there must exist a run $r_{P_i} = (v(0), \widehat{q}(0))(v(1), \widehat{q}(1)) \cdots (v(n+1), \widehat{q}(n+1))$, which is accepted by P_i, and generate word $w_{\widehat{A}_i}$. Hence, we have $w_{\widehat{A}_i} \in \mathcal{L}(P_i)$. ■

Next, we obtain A_i^E that accepts $\mathcal{L}(P_i)$ by removing the environment information that is stored in P_i. To achieve this, we collapse the states of P_i by taking ϵ-closure, determinizing, and minimizing P_i. See [22] for more details about these standard procedures. An example that shows the construction of A_i^E, given T_i and A_i, is illustrated in Fig. 5. Given a word $w \in$

$\mathcal{L}(A_i^E)$, there exists a word $w' \in \mathcal{L}(P_i)$, such that $w'\restriction_{\Sigma_i} = w$. By the use of this fact, the following proposition shows that A_i^E captures the largest subset of the language that is accepted by A_i which can be implemented by the robot $\mathcal{A}_i$ in the environment.

Proposition 5.3: A word $s_i \in \mathcal{L}(A_i)$, $i \in I$ can be used to generate an MS plan ms_i for $\mathcal{A}_i$, such that $ms_i\restriction_{\Sigma_i} = s_i$, if and only if $s_i \in \mathcal{L}(A_i^E)$.

Proof "$\Longleftarrow$": We propose the following three-step procedure to construct an MS plan ms_i given $s_i \in \mathcal{L}(A_i^E)$:break 1) construct a DFA A_i^s that only accepts s_i; 2) construct $\widehat{A}_i^s$ from A_i^s according to (5), and 3) construct the product automaton, i.e., $P_i^s = T_i \otimes \widehat{A}_i^s$. According to its construction, $\widehat{A}_i^s$ accepts only the words $w_{\widehat{A}_i} \in (\{\epsilon\} \cup \Sigma_i)^*$, such that $w_{\widehat{A}_i}\restriction_{\Sigma_i} = s_i$. Since $s_i \in \mathcal{L}(A_i^E)$, there must exist a trajectory of T_i, satisfying a word $w_{\widehat{A}_i}$ (see Proposition 5.2). Therefore, the language of P_i^s is nonempty. Since $\mathcal{L}(P_i^s) \neq \emptyset$, we can find an accepted run r_{P_i} of P_i^s (this can be achieved by a backward reachability search as described earlier) and the corresponding accepted word, i.e., $w_i = w_i(0) \dots w_i(n)$. We obtain a trajectory, i.e., $r_{T_i} = v_i(0) \dots v_i(n)$, of T_i that satisfies w_i by projecting r_{P_i} onto T_i. Then, we obtain a word, i.e., $w_i' = v_i(0)w_i(0)v_i(1)w_i(1) \dots v_i(n)w_i(n)$, such that $a(w_i(j)) = v_i(j)$, where $j \in \{1, \dots, n\}$, for all $w_i(j) \neq \epsilon$. Finally, we obtain $ms_i = w_i'\restriction_{\Sigma_i \cup V}$. Since $ms_i \in (\Sigma_i \cup V)^*$, ms_i meets all the conditions in Definition 3.1. Therefore, following the procedure outlined earlier, ms_i can always be generated from a word $s_i \in \mathcal{L}(A_i^E)$, and ms_i is an MS plan for the robot $\mathcal{A}_i$.

"$\Longrightarrow$": If there exists an MS plan ms_i, such that $ms_i\restriction_{\Sigma_i} = s_i$, then there exists a motion plan $m_i = ms_i\restriction_V$ that satisfies a word $w_i \in (\Sigma \cup \epsilon)^*$ and $w_i\restriction_{\Sigma_i} = s_i$. Hence, according to Proposition 5.2, $w_i \in \mathcal{L}(P_i)$, where $P_i = T_i \otimes \widehat{A}_i$. Since A_i^E accepts all the words in $\mathcal{L}(P_i)\restriction_{\Sigma_i}$, then we have $s_i \in \mathcal{L}(A_i^E)$, which completes the proof. ■

Note that the proof of Proposition 5.3 provides a procedure that guarantees to generate an MS plan ms_i, given a word $s_i \in \mathcal{L}(A_i^E)$, $i \in I$, such that s_i is the service plan for ms_i.

Finally, the implementable global behaviors of the team can be modeled by the SP of the implementable local specifications A_i^E, which is defined as follows.

Definition 5.2 (Synchronous Product): The SP of n FSAs, i.e., $A_i^E = (Q_i^E, q_{0_i}^E, \Sigma_i, \rightarrow_i^E, F_i^E)$, which is denoted by $\|_{i=1}^n A_i^E$, is an FSA Reach$((Q_G, q_{0_G}, \Sigma, \rightarrow_G, F_G))$,[1] where $Q_G = Q_1 \times Q_2 \times \dots \times Q_n$, $q_{0_G} = (q_{0_1}, q_{0_2}, \dots, q_{0_n})$, and $F_G = F_1 \times F_2 \times \dots \times F_n$. The transition relation $\rightarrow_G \subseteq Q_G \times \Sigma \times Q_G$ is defined by $q \xrightarrow{\sigma}_G q'$ if and only if $\forall\ i \in I_\sigma : q[i] \xrightarrow{\sigma}_i^E q'[i]$ and $\forall\ i \notin I_\sigma : q[i] = q'[i]$, where $q[i]$ denotes the ith component of q.

Case Study 1 *(Revisited):* Returning to the proposed example, we first construct A_1^E and A_2^E and, then, the SP $A_1^E \parallel A_2^E$. Since RULE is fully connected, all the words that are accepted by A_i can be implemented. The constructed FSAs are shown in Fig. 6.

[1]For an FSA A let Reach(A) denote the automaton that is obtained by keeping only the states and the transitions from A that are reachable from the initial state q_0.

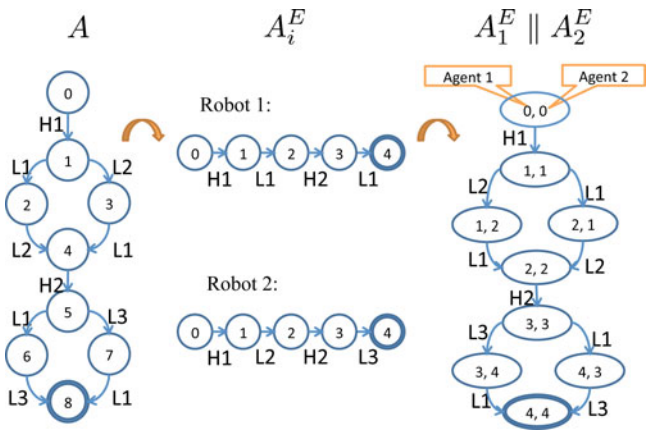


Fig. 6. FSAs generated in case study 1.

B. Synthesis of Individual Motion and Service Plans

To solve Problem 3.1, we need to find a satisfying set of MS plans. Specifically, we aim to find a set of service plans $\{s_i, i \in I\}$, such that $\|_{i\in I}\ \{s_i\} \subseteq \mathcal{L}(A)$ and $\|_{i\in I}\ \{s_i\} \neq \emptyset$. First, we make the important observation that a trace-closed specification is sufficient to satisfy this requirement and provide a solution to Problem 3.1. Formally, we have the following.

Proposition 5.4: Given a language L and a distribution, i.e., $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$ of Σ, if L is a trace-closed language over Δ and $w \in L$, then $\|_{i\in I}\ \{w\restriction_{\Sigma_i}\} \subseteq L$.

Proof: We first prove the following statement. Given a distribution $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$ of Σ and a word $w \in \Sigma^*$, we have $[w]_\Delta = \|_{i\in I}\ \{w\restriction_{\Sigma_i}\}$. For all words $w' \in [w]_\Delta$, according to Definition 2.5, $w'\restriction_{\Sigma_i} = w\restriction_{\Sigma_i} \forall i \in I$. According to Definition 2.4, since $w' \in \Sigma^*$ and $w'\restriction_{\Sigma_i} = w\restriction_{\Sigma_i} \forall i \in I$, then $w' \in \|_{i\in I}\ \{w\restriction_{\Sigma_i}\}$. Hence, $[w]_\Delta \subseteq \|_{i\in I}\ \{w\restriction_{\Sigma_i}\}$. For all words $w' \in \|_{i\in I}\ \{w\restriction_{\Sigma_i}\}$, according to Definition 2.4, $w\restriction_{\Sigma_i} = w'\restriction_{\Sigma_i}$. According to Definition 2.5, $w' \sim_\Delta w$, which implies $w' \in [w]_\Delta$. Hence, $\|_{i\in I}\ \{w\restriction_{\Sigma_i}\} \subseteq [w]_\Delta$. Combined with the fact that $[w]_\Delta \subseteq \|_{i\in I}\ \{w\restriction_{\Sigma_i}\}$, we have $[w]_\Delta = \|_{i\in I}\ \{w\restriction_{\Sigma_i}\}$.

According to Definition 2.5, we have $[w]_\Delta \subseteq L$ for all $w \in L$. Since $[w]_\Delta = \|_{i\in I}\ \{w\restriction_{\Sigma_i}\}$, we have $\|_{i\in I}\ \{w\restriction_{\Sigma_i}\} \subseteq L$ for all $w \in L$. Therefore, the proof is complete. ■

Case Study 1 *(Revisited):* The language that satisfies ϕ [see (10)] is trace closed over the given distribution since all of its words

$$H_1L_1L_2H_2L_1L_3, \quad H_1L_1L_2H_2L_3L_1$$
$$H_1L_2L_1H_2L_3L_1, \quad H_1L_2L_1H_2L_1L_3$$

are trace equivalent. By the projection of $w = H_1L_1L_2H_2L_1L_3$ on the given distribution, we obtain $w\restriction_{\Sigma_1} = H_1L_1H_2L_1$ and $w\restriction_{\Sigma_2} = H_1L_2H_2L_3$, where $\|_{i\in I}\ \{w\restriction_{\Sigma_i}\}$ satisfies L_ϕ. On the other hand, the specification $H_1L_1L_2H_2L_1L_3$ by itself is not trace closed since its trace-equivalent word $H_1L_2L_1H_2L_1L_3$ violates the specification. This is intuitive, since L_1 and L_2 are independent and can be executed in parallel. We cannot find a distributed solution for this specification, since the parallel execution might produce a "wrong" order of serviced requests, violating the specification.

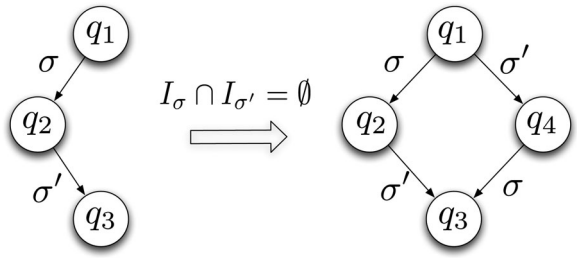


Fig. 7. Independent diamond property.

Our approach aims to construct a DFA A_G whose language is both trace closed and included in $\mathcal{L}(A)$. By Proposition 5.4, an arbitrary word that is accepted by A_G can be used to generate a set of service plans that satisfies the desired requirement by the projection of this word onto the given distribution Δ. Furthermore, we need to guarantee that the word in $\mathcal{L}(A_G)$ can be implemented by the team of robots. To generate $\mathcal{L}(A_G)$, we produce the intersection of the trace-closed subset of $\mathcal{L}(A)$ and the implementable global behaviors of the team $\mathcal{L}(\|_{i\in I}\ A_i^E)$. The intersections of regular languages can be produced by taking products of automata.[2]

To find A_G, we first check if $\mathcal{L}(A)$ is trace closed. An algorithm that checks this property for an arbitrary DFA A is summarized in Algorithm 1. Specifically, we can check if $\mathcal{L}(A)$ is trace closed because of the following result from [24]. Given a distribution Δ of Σ and a minimal DFA A, $\mathcal{L}(A)$ is trace closed if and only if A satisfies the independent diamond (ID) property. The ID property is illustrated in Fig. 7 and defined as the following.

Definition 5.3 (Independent Diamond Property): Given a distribution, i.e., $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$, of Σ and a minimal DFA, i.e., $A = (Q, q_0, \Sigma, \rightarrow_A, F)$, we say that the DFA satisfies the ID property if for any $q_1, q_2, q_3 \in Q$ and $\sigma, \sigma' \in \Sigma$, we have

$$q_1 {}^{\sigma}\!\!\rightarrow q_2 \stackrel{\sigma'}{\rightarrow} q_3 \wedge (I_\sigma \cap I_{\sigma'} = \emptyset) \Rightarrow$$
$$\exists q_4 \in Q \text{ such that } q_1 \stackrel{\sigma'}{\rightarrow} q_4 \stackrel{\sigma}{\rightarrow} q_3. \quad (6)$$

If $\mathcal{L}(A)$ is trace closed, we define $A_G = A \times \|_{i\in I}\ A_i^E$. Otherwise, we define $A_G = \neg(\|_{i\in I}\ B_i) \times \|_{i\in I}\ A_i^E$, where $B_i = B{\restriction}_{\Sigma_i}$ and $B = \|_{i\in I}\ A_i^E \times (\neg A)$. In the second case, A_G is constructed to remove words $w \in \mathcal{L}(\|_{i\in I}\ A_i^E)$ that cannot be used to generate desired individual service plans for the robots [i.e., $\|_{i\in I}\ \{s_i = w{\restriction}_{\Sigma_i}\} \notin \mathcal{L}(A)$]. The following proposition shows that A_G satisfies the desired requirement in both cases.

Proposition 5.5: $\mathcal{L}(A_G)$ is a trace-closed language over Δ and $\mathcal{L}(A_G) \subseteq \mathcal{L}(A)$.

Proof: If $\mathcal{L}(A)$ is trace closed, then $\mathcal{L}(A_G) = \mathcal{L}(A) \cap \mathcal{L}(\|_{i\in I}\ A_i^E)$. Hence, $\mathcal{L}(A_G) \subseteq \mathcal{L}(A)$. Since the language of an SP is a product language that is always trace closed, then $\mathcal{L}(\|_{i\in I}\ A_i^E)$ is trace closed. Since $\mathcal{L}(A)$ is trace closed and the class of trace-closed language is closed under intersection, $\mathcal{L}(A_G)$ is also trace closed.

If $\mathcal{L}(A)$ is not trace closed, then $\mathcal{L}(A_G) = \overline{\mathcal{L}(\|_{i\in I}\ B_i)} \cap \mathcal{L}(\|_{i\in I}\ A_i^E)$. Since $\mathcal{L}(\|_{i\in I}\ B_i)$ and $\mathcal{L}(\|_{i\in I}\ A_i^E)$ are both product languages, then they are both trace closed. Since trace-closed languages are closed under complementation and intersection, $\mathcal{L}(A_G)$ is also a trace-closed language. Since $\mathcal{L}(B) = \mathcal{L}(\|_{i\in I}\ A_i^E) \cap \overline{\mathcal{L}(A)}$, then $\mathcal{L}(\|_{i\in I}\ A_i^E) = \mathcal{L}(B) \cup (\mathcal{L}(\|_{i\in I}\ A_i^E) \cap \mathcal{L}(A))$. Hence, $\mathcal{L}(A_G) = \overline{\mathcal{L}(\|_{i\in I}\ B_i)} \cap (\ \mathcal{L}(B) \cup (\mathcal{L}(\ \|_{i\in I}\ A_i^E) \cap \mathcal{L}\ (A))) = (\overline{\mathcal{L}(\|_{i\in I}\ B_i)} \cap \mathcal{L}(B)) \cup (\overline{\mathcal{L}(\|_{i\in I}\ B_i)} \cap \mathcal{L}(\|_{i\in I}\ A_i^E) \cap \mathcal{L}(A))$. Since $\mathcal{L}(B) \subseteq \mathcal{L}(\|_{i\in I}\ B_i)$, then $\overline{\mathcal{L}(\|_{i\in I}\ B_i)} \subseteq \overline{\mathcal{L}(B)}$. Hence, $(\overline{\mathcal{L}(\|_{i\in I}\ B_i)} \cap \mathcal{L}(B)) \subseteq (\overline{\mathcal{L}(B)} \cap \mathcal{L}(B)) = \emptyset$. Since $\mathcal{L}(A_G) = \overline{\mathcal{L}(\|_{i\in I}\ B_i)} \cap \mathcal{L}(\|_{i\in I}\ A_i^E) \cap \mathcal{L}(A)$, then $\mathcal{L}(A_G) \subseteq \mathcal{L}(A)$, which completes the proof. ■

Algorithm 1 : Check for trace-closedness

Input: A minimal DFA $A = (Q, q_0, \Sigma, \rightarrow, F)$ and a distribution $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$ over Σ
Output: Whether or not $\mathcal{L}(A)$ is trace-closed

1: **for all** $q \in Q$ **do**
2: Construct $N_q = \{q' \in Q \mid \exists q \stackrel{\sigma}{\rightarrow} q' \text{ and } q \neq q'\}$
3: **for all** $q' \in N_q$ **do**
4: Construct $A_{q'}^q = \{\sigma \in \Sigma \mid q \stackrel{\sigma}{\rightarrow} q'\}$
5: **end for**
6: **end for**
7: **for all** $q_1 \in Q$, $q_2 \in N_{q_1}$ and $q_3 \in N_{q_2}$ **do**
8: **for all** $\sigma \in A_{q_2}^{q_1}$ and $\sigma' \in A_{q_3}^{q_2}$ **do**
9: **if** $I_\sigma \cap I_{\sigma'} = \emptyset$ and $\nexists q_4 \in N_{q_1}$ *s.t.* $\sigma' \in A_{q_4}^{q_1}, \sigma \in A_{q_3}^{q_4}$
10: **return** $\mathcal{L}(A)$ is not trace-closed
11: **end for**
12: **end for**
13: **return** $\mathcal{L}(A)$ is trace-closed

If $\mathcal{L}(A_G)$ is not empty, then a solution to Problem 3.1 can be found by picking any accepted word of A_G. We obtain an accepted word $w_g \in \mathcal{L}(A_G)$ by using a backward reachability search that starts from the set of accepting states and that ends at the initial state. Once obtained, w_g is projected onto the given distribution Δ to generate a set of MS plans by the use of the procedure outlined in the proof of Proposition 5.3.

The overall approach that is proposed in this section is summarized in Algorithm 2. In the next theorem, we show that the solution obtained by Algorithm 2 is provably correct.

Theorem 5.1: If $\mathcal{L}(A_G) \neq \emptyset$, then Algorithm 2 returns a solution to Problem 3.1, i.e., , a set of MS plans $\{ms_i, i \in I\}$ such that $L_{\text{MS}}^{\text{team}} \subseteq L_\phi$ and $L_{\text{MS}}^{\text{team}} \neq \emptyset$.

Proof: If $\mathcal{L}(A_G) \neq \emptyset$, then we can obtain $w_g \in \mathcal{L}(A_G)$.

1) Since $\mathcal{L}(A_G) \subseteq \mathcal{L}(\|_{i\in I}\ A_i^E)$, the word $w_g \in \mathcal{L}(\|_{i\in I}\ A_i^E)$. Hence, $s_i \in \mathcal{L}(A_i^E)$. Steps 17–21 in Algorithm 2 correspond to the procedure that is described in the proof of Proposition 5.3. According to Proposition 5.3, a set of MS plans $\{ms_i, i \in I\}$ can always be generated by the set of words $\{s_i, i \in I\}$, such that $ms_i{\restriction}_{\Sigma_i} = s_i$ for all $i \in I$.
2) According to the construction of $\{ms_i, i \in I\}$, $s_i = ms_i{\restriction}_{\Sigma_i} = w_g{\restriction}_{\Sigma_i}$. According to Proposition 5.5, $\mathcal{L}(A_G)$ is trace closed, and $\mathcal{L}(A_G) \subseteq \mathcal{L}(A)$. Since $w_g \in \mathcal{L}(A_G)$, according to Proposition 5.4, $\|_{i\in I}\ \{w_g{\restriction}_{\Sigma_i}\} \subseteq \mathcal{L}(A_G)$.

[2]As a particular case of Definition 5.2, in the case when $n = 2, \Sigma_1 = \Sigma_2 = \Sigma$, and A_1 and A_2 are DFAs, the SP $\|_{i=1}^2\ A_i$ is called simply the product of automata A_1 and A_2 and is denoted by $A_1 \times A_2$, where $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$ [27]. Consequently, we can use products of automata to obtain intersections of regular languages.

Algorithm 2 : Obtain a satisfying set of MS plans from a global specification ϕ

Input: A RE ϕ over Σ, a distribution $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$ of Σ, and a set of TS's $\{T_i = (V, v_{0_i}, \rightarrow_i, \Pi, \models_i),\ i \in I\}$

Output: A set of MS plans $\{ms_i, i \in I\}$

1: Convert ϕ to a minimal DFA A and construct $\{A_i, i \in I\}$
2: Construct $\{\widehat{A}_i, i \in I\}$ and $\{P_i = \widehat{A}_i \otimes T_i, i \in I\}$
3: Take ϵ-closure, determinize, and minimize P_i to obtain $\{A_i^E, i \in I\}$
4: Construct the synchronous product $\|_{i\in I}\ A_i^E$
5: **if** $\mathcal{L}(\|_{i\in I}\ A_i^E) = \emptyset$ **then**
6: **return** no solution exists
7: **else**
8: Check if $\mathcal{L}(A)$ is trace-closed using Alg. 1
9: **if** $\mathcal{L}(A)$ is trace-closed **then**
10: Construct $A_G = A \times \|_{i\in I}\ A_i^E$
11: **else**
12: $A_G = \neg(\|_{i\in I}\ ((\|_{i\in I}\ A_i^E \times (\neg A)) \upharpoonright_{\Sigma_i})) \times \|_{i\in I}\ A_i^E$
13: **end if**
14: **if** $\mathcal{L}(A_G) = \emptyset$ **then**
15: **return** no solution found
16: **else**
17: Find a word $w_g \in \mathcal{L}(A_G)$ and obtain a set of local words $s_i = w_g \upharpoonright_{\Sigma_i}, i \in I$
18: **for** all $i \in I$ **do**
19: Construct $\widehat{A}_i^s$ from A_i^s, where $\mathcal{L}(A_i^s) = s_i$
20: Construct $P_i^s = \widehat{A}_i^s \otimes T_i$ and find an accepted run $r_{P_i^s}$ and the corresponding accepted word $w_i = w_i(0) \ldots w_i(n)$.
21: Obtain $r_{T_i} = \gamma_{T_i}(r_{P_i^s}) = v_i(0) \ldots v_i(n+1)$ and $ms_i = v_i(0)w_i(0) \ldots v_i(n)w_i(n) \upharpoonright_{V \cup \Sigma_i}$
22: **end for**
23: **return** $\{ms_i, i \in I\}$
24: **end if**
25: **end if**

Hence, $\|_{i\in I}\ \{s_i\} \subseteq \mathcal{L}(A)$. Since $L_{\mathrm{MS}}^{\mathrm{team}} = \|_{i\in I}\ \{s_i\}$, we have $L_{\mathrm{MS}}^{\mathrm{team}} \subseteq \mathcal{L}(A)$. Since $\mathcal{L}(A) = L_\phi$, we have $L_{\mathrm{MS}}^{\mathrm{team}} \subseteq L_\phi$.

3) By construction of $\{ms_i, i \in I\}$, $s_i = w_g \upharpoonright_{\Sigma_i}$; therefore, $w_g \in \|_{i\in I}\ \{s_i\}$. Hence, $L_{\mathrm{MS}}^{\mathrm{team}} \neq \emptyset$.

In the rest of this section, we discuss the completeness of the approach.

Proposition 5.6: If $\mathcal{L}(A)$ is trace closed over Δ, then Algorithm 2 returns a solution to Problem 3.1 if one exists.

Proof: If $\mathcal{L}(A)$ is trace closed over Δ, we have $A_G = A \times \|_{i\in I}\ A_i^E$. Assume that there is a solution to Problem 3.1, which means that there is a set of MS plans $\{ms_i, i \in I\}$ such that the corresponding set of service plans $\{s_i, i \in I\}$ satisfies $\|_{i\in I}\ \{s_i\} \subseteq \mathcal{L}(A)$ and $\|_{i\in I}\ \{s_i\} \neq \emptyset$. According to Proposition 5.1, $s_i \in (A_i)$. According to Proposition 5.3, $s_i \in (A_i^E)$. Hence, $\|_{i\in I}\ \{s_i\} \subseteq \mathcal{L}(\|\ i \in I(A_i^E))$. Since $A_G = A \times \|_{i\in I}\ A_i^E$, $\|_{i\in I}\ \{s_i\} \subseteq \mathcal{L}(\|\ i \in I(A_i^E))$, and $\|_{i\in I}\ \{s_i\} \subseteq \mathcal{L}(A)$, we have $\|_{i\in I}\ \{s_i\} \subseteq \mathcal{L}(A_G)$. Since $\|_{i\in I}\ \{s_i\} \neq \emptyset$, we have $L(A_G) \neq \emptyset$. According to Theorem 5.1, Algorithm 2 returns a solution to Problem 3.1. The proof is complete. ∎

If $\mathcal{L}(A)$ is not trace closed, a complete solution to Problem 3.1 requires to find a nonempty trace-closed subset of $\mathcal{L}(A)$ if one exists. Equivalently, we can formulate it as the problem of finding $\mathcal{L}(A)_\Delta$, given $\mathcal{L}(A)$ and Δ. We show in the next proposition that this problem is undecidable. Therefore, if $\mathcal{L}(A)$ is not trace closed, our approach to Problem 3.1 is not complete, and there exists no general solution to the problem.

Proposition 5.7: The problem of finding a nonempty trace-closed subset of a regular language L is undecidable.

The undecidability is proved using a reduction to Post's correspondence problem (PCP), which is known to be undecidable [28]. We skip the details and only mention that this is an adaptation of a proof in [24], which in turn is based on a construction from [29].

Case Study 1 *(Revisited):* By the application of Algorithm 1, we verify that L_ϕ is trace closed since its corresponding minimal DFA A that is shown in Fig. 6 satisfies the ID property. Thus, we have $A_G = A \times (A_1^E \parallel A_2^E)$. We choose $w_g = H_1 L_1 L_2 H_2 L_1 L_3 \in \mathcal{L}(A_G)$. The corresponding service plans for the two robots are $s_1 = H_1 L_1 H_2 L_1$ and $s_2 = H_1 L_2 H_2 L_3$, respectively.

VI. Complexity

In this section, we analyze the computational complexity of the algorithms that are proposed in Section V, given the assumption that a request does not occur in more than one vertex. The running time of Algorithm 1 (i.e., to check if a language of a minimal DFA, i.e., $A = (Q, q_0, \Sigma, \rightarrow, F)$, is trace closed) is bounded above by $O(|Q| \cdot |\Sigma|)$. The running time of Algorithm 2 depends essentially on the construction of A_G. Furthermore, the construction of A_G relies primarily on the construction of A_i^E and $\|_{i\in I}\ A_i^E$, which maps to step 3 and 4 in Algorithm 2. In the rest of the section, we discuss, in more detail, the size of A_i^E and $\|_{i\in I}\ A_i^E$, as well as the running time of steps 3 and 4. We denote $|A|$ as the number of states in A, if A is an FSA.

Proposition 6.1: $|A_i^E|$ and $|\ \|_{i\in I}\ A_i^E|$ are bounded above by $|\ \rightarrow_{A_i}\ |$ and $\prod_{i\in I} |\ \rightarrow_{A_i}\ |$, respectively.

Proof: To prove Proposition 6.1, we first prove the following statement. The number of states in the DFA, which is denoted as A_i^D and obtained by taking ϵ-closure and determinizing the NFA, i.e., $P_i = T_i \otimes \widehat{A}_i$, is bounded above by the number of transitions $|\ \rightarrow_{A_i}\ |$ in the DFA A_i.

See [22] for details of the subset construction algorithm for ϵ-closure and determinization. Via this algorithm, an equivalent DFA is constructed from an NFA by the generation of subsets of the states of the NFA, which then become the states of the equivalent DFA.

We first prove by contradiction that for each subset of Q_{P_i} (i.e., a new state in A_i^D) constructed during the subset construction algorithm, all states (v, q) in this subset have the same second component $q \in Q_i$. If this is not the case, then if there exist two states (v, q) and (v, q') in the same subset and $q \neq q'$, we can reach both (v, q) and (v, q') from the initial state, given the same sequence of inputs. Thus, by the construction of P_i, we can reach q and q' from the initial state of q_{0_i} given the same sequence of inputs. However, this contradicts with the fact that A_i is a DFA. Therefore, we have that all states (v, q) in each subset of Q_{P_i} have the same second component q.

For each state $q \in Q_i$, we denote S_i^q as the set of states $\{(v,q) \in Q_{P_i}, v \in V\}$. From the previous paragraph, we know that all the subsets that we constructed during the subset construction algorithm are in fact the subsets of S_i^q, $q \in Q_i$. For each S_i^q, we denote Σ_i^q as the set of requests $\{\sigma \in \Sigma_i \mid (v',q') \xrightarrow{\sigma}_{P_i} (v,q), (v',q') \in Q_{P_i}$ and $(v,q) \in S_i^q\}$.

Now, we show that given $q \in Q_i$, the number of subsets of S_i^q that can be constructed during the subset construction algorithm is bounded above by $|\Sigma_i^q|$. If $(v_1,q') \xrightarrow{\sigma}_{P_i} (v_3,q)$ and $(v_2,q'') \xrightarrow{\sigma}_{P_i} (v_4,q)$, where (v_3,q) and $(v_4,q) \in S_i^q$ and $v_3 \neq v_4$, then $v_1 = v_2 = v$ since σ can occur at only one vertex (i.e., $a(\sigma) = v$), and $(v,q') \xrightarrow{\sigma}_{P_i} (v_4,q)$ and $(v,q'') \xrightarrow{\sigma}_{P_i} (v_3,q)$. This is trivially true if $v_3 = v_4$. Hence, (v_1,q') and (v_2,q'') with the same input σ must reach the same set of states, i.e., $N_i^{\sigma,q} = \{(v,q) \in S_i^q \mid a(\sigma) = v', (v',v) \in \rightarrow_i\}$. According to the construction of P_i, for all transitions $(v,q) \xrightarrow{\epsilon}_{P_i} (v',q')$, we have $q = q'$. After taking ϵ-closure of $N_i^{\sigma,q}$, we obtain a subset of S_i^q, which is denoted as $S_i^{\sigma,q} = \{(v,q) \mid v \in \text{Reach}(\sigma)\}$, where $\text{Reach}(\sigma)$ is the set of vertices that can be reached from the vertex $a(\sigma)$. Since $(v',q') \xrightarrow{\sigma}_{P_i} (v,q)$ only if $q' \xrightarrow{\sigma}_{A_i} q$ and $v' = a(\sigma)$, then all states (v',q') taking the input sequence $\sigma\epsilon^*$ always reach the same subset $S_i^{\sigma,q}$. For each q, since each subset containing a state (note that there can be at most 1), which can take input $\sigma\epsilon^*$ always reaches the same subset of S_i^q, the number of constructed subsets of S_i^q is smaller than or equal to $|\Sigma_i^q|$.

Finally, since the number of constructed subsets of Q_{P_i} is smaller than or equal to $\sum_{q \in Q_i} |\Sigma_i^q|$, which is smaller than or equal to $|\rightarrow_{A_i}|$, the statement is proved.

Following from the statement that we just proved, the construction of A_i^E (A_i^E is obtained by the minimization of A_i^D) and the definition of the SP (see Definition 5.2), we see that the number of states in A_i^E and $\|_{i \in I} A_i^E$ are bounded above by $|\rightarrow_{A_i}|$ and $\prod_{i \in I} |\rightarrow_{A_i}|$, respectively. ■

Proposition 6.2: The running time to construct A_i^E (step 3 in Algorithm 2) is bounded above by $O(|\rightarrow_{A_i}| \cdot |V|) + O(|\rightarrow_{A_i}| \cdot \log |\rightarrow_{A_i}|)$, and the running time to construct $\|_{i \in I} A_i^E$ (step 4 in Algorithm 2) is bounded above by $O((\prod_{i \in I} |\rightarrow_{A_i}|)^2 \cdot |\Sigma|)$.

Proof: To prove the first part of Proposition 6.2, we first prove that the complexity of construction of A_i^D (we use the same notation A_i^D as in the proof of Proposition 6.1) is bounded above by $O(|\rightarrow_{A_i}| \cdot |V|)$.

As shown in the proof of Proposition 6.1, the number of constructed subsets of Q_{P_i} (i.e., the states of A_i^D) is smaller than or equal to $|\rightarrow_{A_i}|$. According to the subset construction algorithm, the complexity of construction of a new subset that can be reached from a set of states is linear in the number of states in S_i^q. (We use the same definition as in the proof of Proposition 6.1.) Note that $|S_i^q| = |V|$. Therefore, $O(|\rightarrow_{A_i}| \cdot |V|)$ is the upper bound of the complexity of taking ϵ-closure and determinizing the FSA P_i.

By the usage of the minimization algorithm that is described in [22], the running time of minimization of the DFA A_i^D is linear in $n \log n$, where $n = |A_i^D|$. Since we obtain A_i^E by the construction of A_i^D and, then, minimization of A_i^D, the first part of Proposition 6.2 is proved.

To construct the SP of FSAs, we first generate the set of states of $\|_{i \in I} A_i^E$ by taking the Cartesian product of Q_i^E, $i \in I$, where Q_i^E represents the set of states of A_i^E. Then, we check if there exist transitions between each pair of states of $\|_{i \in I} A_i^E$. Hence, the running time to construct $\|_{i \in I} A_i^E$ is bounded above by $O((\prod_{i \in I} |\rightarrow_{A_i}|)^2 \cdot |\Sigma|)$. Therefore, the proof is complete. ■

According to the construction of A_G (see Section V-B), if $\mathcal{L}(A)$ is trace closed, then $|A_G|$ (constructed in step 10) is at most $|A| \cdot |\|_{i \in I} A_i^E|$. Otherwise, $|A_G|$ (constructed in step 12) is at most $|\|_{i \in I} B_i| \cdot |\|_{i \in I} A_i^E|$.

Remark 6.1: Note that $|A_G|$ is not related to the size of the TS T_i but only with A_i, which is apparent from Proposition 6.1 and the fact that the size of B_i and $\|_{i \in I} B_i$ depend only on A and the distribution Δ. This fact substantiates the statement made in Section I that we avoid the construction of the parallel composition of the individual motions (represented by T_i) and prevent state-space explosions.

VII. Relaxing the Simplifying Assumptions in Problem 3.1

There were two simplifying assumptions made in the formulation of Problem 3.1 and its solution described in the previous sections: 1) No two vertices can share a request, and 2) the robots can communicate with each other only when they are at the same vertex. As stated in Section III, these assumptions were made for simplicity of notation and to reduce the complexity of the overall approach. However, these assumptions may be restrictive from a practical point of view. One can imagine scenarios, where the same request occurs at several different vertices (e.g., data are available at different locations). Furthermore, by using wireless or other types of communication devices, the robots can possibly communicate and, therefore, cooperate to service requests at different vertices (regions) in the graph (environment).

To relax the first assumption, we now model the locations of the requests as a function $a : \Sigma \rightarrow 2^V$ (as opposed to a function that takes values in V as before) with the following semantics: $v \in a(\sigma)$ means that service request σ occurs at the vertex v. If a request σ occurs at different vertices in the environment (i.e., $|a(\sigma)| > 1$), then we say that σ is serviced if there exists a time instant at which all the robots that own σ are at vertices, where σ occurs (two or more robots are allowed to overlap at a vertex). A practical example of a request that occurs at multiple vertices is the case in which an agent has several options to collect a certain piece of data. This also allows for a situation in which the agents that own σ need to synchronize to make a collective decision, possibly based on information they collected individually earlier.

The definition of the TS T_i, $i \in I$, [see (2)] remains the same, with the exception of the satisfaction relation $\models_i \subseteq V \times \Pi$, which is redefined as follows: $(v,\epsilon) \in \models_i$ for all $v \in V$ and $(v,\sigma) \in \models_i$, $\sigma \in \Sigma_i$, if and only if $v \in a(\sigma)$. In other words, all requests that occur at vertex v become the observations of the state v of T_i.

To relax the second assumption, we model the communication capabilities of the robots as an undirected communication graph

$$\mathbb{C} = (V, E_{\mathbb{C}}) \tag{7}$$

where $E_{\mathbb{C}} \subseteq V \times V$ is a symmetric relation that models the environment-induced interrobot communication constraints. Specifically, $(v_i, v_j) \in E_{\mathbb{C}}$ if and only if a robot that is located at v_i can directly communicate with another robot that is located at v_j. We use $\mathbb{C}_k = (V_k, E_k)$, where $k \in K$, $V_k \subseteq V$, and $E_k \subseteq E$, to denote a connected component (CC) of an undirected graph (a CC is a maximal connected subgraph of an undirected graph). K is a set that indexes all CCs of an undirected graph. A partition of the set V can be obtained from the collection of subsets $\{V_k, k \in K\}$, where V_k is the set of vertices of $\mathbb{C}_k$ and $\cup_{k \in K} V_k = V$. We say that two robots can communicate with each other if they locate in the same CC. According to the semantics of servicing requests as given earlier, in order to service a shared request σ, all robots that own σ must be at vertices, where σ occurs at the same time, and be part of the same CC.

With the two relaxed assumptions as described earlier, Problem 3.1 can be reformulated as follows.

Problem 7.1: Given a team of agents $\mathcal{A}_i$, $i \in I$, with motion capabilities T_i [see (2) with $\models_i$ adapted as described earlier] and communication constraints $\mathbb{C}$ [see (7)] on a graph $\mathcal{E}$ [see (1)], a set of requests Σ, a function $a : \Sigma \to 2^V$ that represents the locations of the requests, a distribution, i.e., $\Delta = \{\Sigma_i \subseteq \Sigma, i \in I\}$, of Σ that models the capacity of the robots to service requests and the cooperation requirements among the robots, and a task specification ϕ in the form of an RE over Σ find a set of MS plans $\{ms_i, i \in I\}$ such that the motion of the team satisfies ϕ.

We first consider the particular case when for all shared requests σ, the vertices in the set $a(\sigma) \subseteq V$ are connected in the graph $\mathbb{C}$. In this case, all robots that own a shared request σ can always communicate with each other and service σ simultaneously when they visit vertices in $a(\sigma)$. Problem 7.1 is then reduced to Problem 3.1 with a relaxed assumption for the location of the requests, which can be viewed as Problem 3.1 with a modified function a and relation $\models_i$ of T_i. Note that in the approach that is outlined in Section V, $\models_i$ is only used in the definition of the product automaton P_i (see Definition 5.1). Since Definition 5.1 also applies to the modified $\models_i$, the previous approach can be used to solve this special case of Problem 7.1 without any changes, and all the results that are shown in Section V still hold.

Next, we show that the general case of Problem 7.1 can be solved by reducing it to the special case described earlier. Specifically, we treat a shared request σ that occurs in different CCs as different shared requests by labeling σ with the CC $\mathbb{C}_k$. We denote $\sigma_{\mathbb{C}_k}$ as the relabeling of σ in the CC $\mathbb{C}_k$. For the given set of requests Σ, the distribution Δ, the task specification ϕ, and communication graph $\mathbb{C}$ for Problem 7.1, we then construct the following:

1) a set of requests
$$\Sigma^{\mathbb{C}} = \{\sigma \mid |I_\sigma| = 1\} \cup \bigcup_{\substack{k \in K, \sigma \in \Sigma \\ |I_\sigma| > 1}} \sigma_{\mathbb{C}_k};$$
2) a distribution $\Delta^{\mathbb{C}} = \{\Sigma_i^{\mathbb{C}} \subseteq \Sigma^{\mathbb{C}}, i \in I\}$ such that 1) for all $\sigma \in \Sigma$, we have $\sigma \in \Sigma_i^{\mathbb{C}}$ if and only if $\sigma \in \Sigma_i$ and 2) for all $\sigma_{\mathbb{C}_k} \notin \Sigma$, we have $\sigma_{\mathbb{C}_k} \in \Sigma_i^{\mathbb{C}}$ if and only if the corresponding request $\sigma \in \Sigma_i$ (i.e., if a robot $\mathcal{A}_i$ owns the shared request σ, then $\mathcal{A}_i$ also owns $\sigma_{\mathbb{C}_k}$, for all $k \in K$);
3) a set of labels $I_\sigma^{\mathbb{C}} = \{i \in I \mid \sigma \in \Sigma_i^{\mathbb{C}}\}$ for each request $\sigma \in \Sigma^{\mathbb{C}}$;
4) a task specification $\phi^{\mathbb{C}}$ by replacement of all instances of the shared requests σ in ϕ by $(\sigma_{\mathbb{C}_1} + \cdots + \sigma_{\mathbb{C}_{|K|}})$;
5) a location relation $a^{\mathbb{C}} : \Sigma^{\mathbb{C}} \to 2^V$ such that 1) for all $\sigma \in \Sigma^{\mathbb{C}} \cap \Sigma$, we have $v \in a^{\mathbb{C}}(\sigma)$ if and only if $v \in a(\sigma)$ and 2) for all $\sigma_{\mathbb{C}_k} \in \Sigma^{\mathbb{C}} \backslash \Sigma$, we have $v \in a^{\mathbb{C}}(\sigma_{\mathbb{C}_k})$ if and only if $v \in V_k$ and the corresponding request $\sigma \in \Sigma$ satisfies $v \in a(\sigma)$;
6) TS $T_i^{\mathbb{C}} = \{V, v_{0_i}, \to_i, \Pi^{\mathbb{C}}, \models_i^{\mathbb{C}}\}$, where $\Pi^{\mathbb{C}} = \Sigma^{\mathbb{C}} \cup \{\epsilon\}$, and $\models_i^{\mathbb{C}} \subseteq V \times \Pi^{\mathbb{C}}$ is a relation, where $(v, \epsilon) \in \models_i^{\mathbb{C}}$ for all $v \in V$, and $(v, \sigma) \in \models_i^{\mathbb{C}}, \sigma \in \Sigma_i^{\mathbb{C}}$, if and only if $v \in a^{\mathbb{C}}(\sigma)$.

By the use of the constructed $\Sigma^{\mathbb{C}}$, $T_i^{\mathbb{C}}$, $a^{\mathbb{C}}$, $\Delta^{\mathbb{C}}$, and $\phi^{\mathbb{C}}$ as inputs of Problem 7.1, we guarantee that for all shared $\sigma \in \Sigma^{\mathbb{C}}$, vertices $a^{\mathbb{C}}(\sigma) \subseteq V$ are connected in the graph $\mathbb{C}$. Hence, the new problem is a special case of Problem 7.1, which means that we can obtain a set of MS plans that satisfies $\phi^{\mathbb{C}}$ by directly using the approach for Problem 3.1. To find the solution to the original problem (i.e., a set of MS plans that satisfies ϕ), we simply replace all the labeled shared requests $\sigma_{\mathbb{C}_k}$ with the corresponding shared requests σ in the obtained MS plans.

Remark 7.1: The computational complexity analysis in Section VI does not apply to our solution for Problem 7.1. The main challenge in the analysis of the complexity to solve Problem 7.1 is to find the upper bound of the size of A_i^E, which now also depends on the occurrence of the requests in the environment and the motion capabilities of the robots. In the worst case, the size of A_i^E is bounded above by the product of the size of A_i and the size of T_i. A better upper bound might be achieved by the consideration of the special structure of the product automaton P_i and will be studied in our future work.

VIII. Automatic Deployment in the Robotic Urban-Like Environment

In our implementation, the global specification ϕ is first converted to the minimal DFA A by using JFLAP [23]. The rest of Algorithm 2 (including Algorithm 1) is implemented in MATLAB. 1) We take a global DFA A, a distribution Δ, and a set of TS T_i as inputs and output a set of individual MS plans for the robotic team. 2) We use Dijkstra's algorithm [30] to find a word or a run that is accepted by an FSA by the assumption that each transition of the FSA has default cost 1; if the algorithm fails to find an accepted run, the language of this FSA is empty. 3) We implement the standard algorithm [22] for taking ϵ-closure, determinizing a ϵ-NFA, and minimizing a DFA. The output of Algorithm 2 is then mapped to control and communication strategies (which is defined in Section III)

through the use of motion primitives and interrupts as described earlier.

In this section, we show how our solution can be used to deploy a team of robots by using a rich specification to service requests that occur in a miniature city. Our RULE (see Figs. 1 and 2) is a collection of roads, intersections, and parking lots, which are connected following a simple set of rules (e.g., a road connects two (not necessarily different) intersections, the parking lots can only be located on the side of (each bound of) a road). Each intersection has traffic lights that are synchronized in the usual way. A desktop computer at 2 GHz and with 2 GB RAM is used to remotely control the traffic lights through XBee wireless boards. Each parking lot consists of several parking spaces, where each parking space can accommodate exactly one car, and each parking lot has enough parking spaces to accommodate all the robots at the same time. The city is easily reconfigurable through retaping and replacement of the wireless traffic lights in intersections.

The robots are Khepera III miniature cars. Each car can sense when entering an intersection from a road, when entering a road from an intersection, when passing in front of a parking lot, when it is correctly parked in a parking space, and when a front obstacle is dangerously close. In particular, the cars can avoid collisions among themselves, which implies that several cars can be in the same region at the same time. Moreover, by ensuring all the cars follow the basic traffic rules and setting reasonable time intervals for the traffic lights, we make sure that motion deadlocks (i.e., two cars fail to move forward because they are blocking each other) do not occur. Each car can distinguish the color of a traffic light and different parking spaces in the same parking lot. Each car is programmed with motion and communication primitives, which allows it to safely drive on a road, turn in an intersection, park, and communicate with other cars. All the cars can communicate through Wi-Fi with the desktop computer that is described earlier, which is used as an interface to the user (i.e., to enter the global specification) and to perform all the computation that is necessary to generate the individual control and communication strategies. Once computed, these are sent to the cars, which execute the task autonomously by interacting with the environment and by communicating with each other, if necessary. We assume that the communication protocol is deadlock free.

As we described in Section III, RULE can be modeled by the usage of the proposed framework. We assume that interrobot communication is possible only when the robots are in the same parking lot. The motion capabilities of the robots are captured by a TS T_i that is illustrated in Fig. 3. Note that, in reality, each vertex of T_i has associated a set of motion primitives, and each transition is triggered by a Boolean combination of interrupts. For example, at vertex R_{5l}, only one motion primitive follow_road is available, which allows the robot to drive on the road. There is only one possible transition from R_{5l} to I_1, which is triggered by at_int AND green_light, where at_int is an interrupt generated when the robot reaches the end of a road at an intersection, and green_light is an interrupt that is generated at the green color of the traffic light. As another example, there are three motion primitives available at I_1, i.e., turn_right_int, turn_left_int, and go_straight_int, which allow the robot to turn right, left, or go straight through an intersection, respectively. The transitions from I_1 to R_{6r}, R_{5r}, R_{3l}, and R_{1r} are all triggered by the same interrupt on_road, which is generated when the robot is back on a road leaving an intersection.

It is important to note that, by the selection of a motion primitive that is available at a vertex, the robot can correctly execute a run of T_i, given that it is initialized on a road. Indeed, only one motion primitive (i.e., follow_road) is available on a road, and at an intersection, the choice of a motion primitive uniquely determines the next vertex given the road that the robot entered the intersection from. For example, by selecting turn_right_int at I_1, the robot goes to R_{1r} given that it came from R_{3r}. This justifies our assumption from Section III that runs of T_i can be executed by the robots. In other words, MS plans that are defined in Section III and derived as described in Section V can be immediately implemented by a robot. It is easy to see that under some reasonable liveness assumptions about environmental events (e.g., the traffic lights will eventually turn green), such a TS captures the motion of each robot correctly. (See [31] for implementation details.)

Assume that two robots, which are labeled as $\mathcal{A}_1$ and $\mathcal{A}_2$, are available for deployment in the city with the topology from Fig. 2. In the rest of this section, we complete the case study that is introduced in the earlier sections and present another case study.

Case Study 1 *(Revisited):* Using Algorithm 2, we generate the MS plans for $\mathcal{A}_1$ and $\mathcal{A}_2$. By the assumption that $\mathcal{A}_1$ and $\mathcal{A}_2$ start in R_{2l} and R_{1l}, respectively, the two MS plans are

$$ms_1 : \begin{array}{l} R_{2l}I_2R_{4r}I_3R_{8r}P_4H_1R_{8r}I_4R_{5l}I_1R_{6r}P_1L_1 \\ R_{6r}I_4R_{8l}P_5H_2R_{8l}I_3R_{8r}I_4R_{5l}I_1R_{6r}P_1L_1 \end{array} \tag{8}$$

$$ms_2 : \begin{array}{c} R_{1l}I_1R_{3l}I_2R_{4r}I_3R_{8r}P_4H_1R_{8r} \\ I_4R_{5l}I_1R_{3l}I_2R_{3r}P_2L_2R_{3r}I_1R_{5r}I_4R_{8l}\,. \\ P_5H_2R_{8l}I_3R_{8r}I_4R_{6l}P_3L_3 \end{array} \tag{9}$$

Snapshots from a movie of the actual deployment are shown in Fig. 8. The movie of the deployment in the RULE platform is available at http://hyness.bu.edu/RULE_media.html.

Case Study 2: Assume $\Sigma = \{H_1, H_2, L_1, L_2, L_3, L_4, L_5\}$, $\Sigma_1 = \{L_1, L_4, H_1, H_2\}$ and $\Sigma_2 = \{L_2, L_3, L_5, H_1, H_2\}$. Consider the following specification: "first service L_4 and then L_5 or first service H_1; both L_1 and L_2 in an arbitrary order; H_2; and finally, both L_1 and L_3 in an arbitrary order." Formally, this specification translates to the following RE over Σ:

$$\phi : \ (L_4L_5 + H_1)\ (L_1L_2 + L_2L_1)\ H_2\ (L_1L_3 + L_3L_1). \tag{10}$$

In this example, $\mathcal{L}(A)$ is not a trace-closed language. Therefore, the FSA A_G is obtained as described in Section V-B. We choose $w_g = H_1L_1L_2H_2L_1L_3 \in \mathcal{L}(A_G)$. The corresponding service plans for $\mathcal{A}_1$ and $\mathcal{A}_2$ are $s_1 = H_1L_1H_2L_1$ and $s_2 = H_1L_2H_2L_3$, respectively. The FSAs that are generated by Algorithm 2 are shown in Fig. 9. Finally, we generate the MS plans for $\mathcal{A}_1$ and $\mathcal{A}_2$ by the assumption that $\mathcal{A}_1$ and $\mathcal{A}_2$ start in R_{2l} and R_{1l}, respectively. Since the service plans and

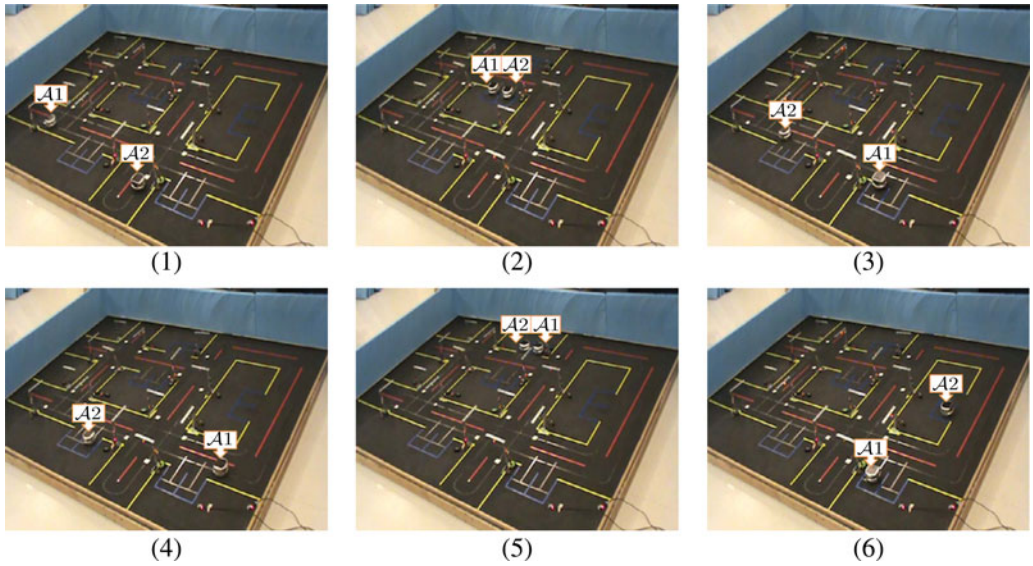


Fig. 8. Six snapshots from the deployment, which correspond to the MSs given in (8) and (9). The labels for the roads, intersections, and parking spaces are given in Fig. 2. (1) Position of the cars immediately after the initial time, when $\mathcal{A}_1$ is on road R_{2l} and $\mathcal{A}_2$ is on road R_{1l}. (2) Two cars visit parking lot P_4 simultaneously to service the "heavy" request H_1. (3) $\mathcal{A}_1$ is in P_1, and therefore, the "light" request L_1 is serviced. (4) $\mathcal{A}_2$ is in P_2, and therefore, request L_2 is serviced. (5) Two cars are in parking lot P_5 to service the "heavy" request H_2. (6) Eventually, $\mathcal{A}_1$ stops in P_1, and $\mathcal{A}_2$ stops in P_3, which means that L_1 and L_3 are serviced.

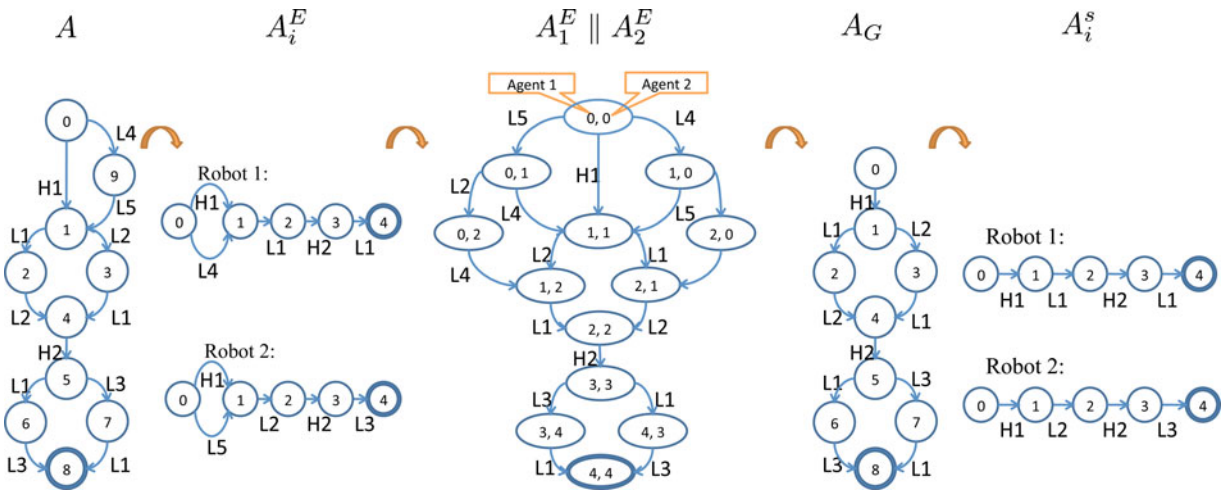


Fig. 9. FSAs generated by the application of Algorithm 2 to case study 2 that is described in Section VIII.

the initial positions of the robots are equal to those in case study 1, we obtain the same MS plans as the ones in case study 1.

IX. Conclusion and Final Remarks

We have presented a framework for automatic deployment of a robotic team from a specification given as an RE over a set of service requests that occur at known locations of a partitioned environment. Given the robot capabilities to service the requests and the possible cooperation requirements for some requests, we have found individual control and communication strategies, such that the global behavior of the team satisfies the given specification. We have illustrated the proposed method with experimental results in our RULE.

The proposed framework does not accommodate for changes in the environment and external events, and it is not robust to agent failures, e.g., loss of communication. As future work, we will study how to re-plan when such changes/events occur. For instance, a reactive approach [32] can be used to accommodate "well-behaved" external events, as recommended in [17]. Moreover, we will study optimal solutions that take into account the motion and service costs. Finally, we will consider extensions of this approach to formulas of temporal logics, such as LTL, and to probabilistic systems, such as Markov decision processes.

Acknowledgment

The authors would like to thank all reviewers for thoughtful comments.

REFERENCES

[1] Y. Chen, X. Ding, A. Stefanescu, and C. Belta, "A formal approach to deployment of robotic teams in an urban-like environment," presented at the Int. Symp. Distrib. Auton. Robot. Syst., Lausanne, Switzerland, Nov. 2010.

[2] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.

[3] J. C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer, 1991.

[4] G. Fainekos, H. Kress-Gazit, and G. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proc. IEEE Conf. Decis. Control Eur. Control Conf.*, Seville, Spain, Dec. 2005, pp. 4885–4890.

[5] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multiagent motion tasks based on LTL specifications," in *Proc. IEEE Conf. Decision Control*, Paradise Islands, The Bahamas, Dec. 2004, pp. 153–158.

[6] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proc. IEEE Conf. Decision Control Chin. Control Conf.*, Shanghai, China, Dec. 2009, pp. 5997–6004.

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

[8] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proc. IEEE*, vol. 88, no. 7, pp. 971–984, Jul. 2000.

[9] R. Milner, *Communication and Concurrency*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[10] 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.

[11] P. Tabuada and G. Pappas, "Linear time logic control of discrete-time linear systems," *IEEE Trans. Automat. Control*, vol. 51, no. 12, pp. 1862–1877, Dec. 2006.

[12] A. Tiwari and G. Khanna, "Series of abstractions for hybrid automata," in *Proc. Int. Conf. Hybrid Systems: Comput. Control* (Lecture Notes in Computer Science Series 2289). Berlin, Germany: Springer-Verlag, 2002, pp. 465–478..

[13] S. Lindemann, I. Hussein, and S. LaValle, "Real time feedback control for nonholonomic mobile robots with obstacles," in *Proc. IEEE Conf. Decision Control*, New Orleans, LA, Dec. 2007, pp. 2406–2411.

[14] S. Karaman and E. Frazzoli, "Vehicle routing with temporal logic specifications: Applications to multi-UAV mission planning," *J. Robust Nonlinear Control*, vol. 21, no. 12, pp. 1372–1395, Aug. 2011.

[15] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Trans. Robot.*, vol. 26, no. 1, pp. 48–61, Feb. 2010.

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

[17] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas, "Courteous cars," *IEEE Robot. Autom. Mag.*, vol. 15, no. 1, pp. 30–38, Mar. 2008.

[18] M. Mukund, *From Global Specifications to Distributed Implementations Norwell*, MA: Kluwer, 2002, pp. 19–34.

[19] Y. Chen, S. Birch, A. Stefanescu, and C. Belta, "A hierarchical approach to automatic deployment of robotic teams with communication constraints," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Taipei, Taiwan, Oct. 2010, pp. 5079–5084.

[20] A. Stefanescu, J. Esparza, and A. Muscholl, "Synthesis of distributed algorithms using asynchronous automata," in *Proc. Int. Conf. Concurrency Theory* (Lecture Notes in Computer Science Series 2761), New York: Springer-Verlag, 2003, pp. 27–41.

[21] M. Karimadini and H. Lin, "Guaranteed global performance through local coordinations," *Automatica*, vol. 47, no. 5, pp. 890–898, 2011.

[22] J. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, 2007.

[23] S. H. Rodger and T. W. Finley, *JFLAP: An Interactive Formal Languages and Automata Package*. Boston, MA: Jones & Bartlett, 2006.

[24] A. Stefanescu, "Automatic synthesis of distributed transition systems," Ph.D. dissertation, Faculty Comput. Sci., Electr. Eng. Inf. Technol., Univ. Stuttgart, Stuttgart, Germany, 2006.

[25] A. Mazurkiewicz, "Introduction to trace theory," in *The Book of Traces*. Singapore: World Scientific, 1995, pp. 3–41.

[26] P. Thiagarajan and J. Henriksen, "Distributed versions of linear time temporal logic: A trace perspective," in *Lectures on Petri Nets I: Basic Models* (Lecture Notes in Computer Science Series 1491). Berlin, Germany: Springer-Verlag, 1998, pp. 643–681.

[27] Y. Sheng, *Regular Languages*. New York: Springer-Verlag, 1997.

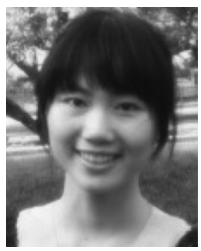
[28] C. H. Papadimitriou, *Computational Complexity*. Reading, MA: Addison-Wesley, 1994.

[29] A. Muscholl and H. Petersen, "A note on the commutative closure of star-free languages," *Inf. Process. Lett.*, vol. 57, no. 2, pp. 71–74, 1996.

[30] T. Cormen, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2001.

[31] M. Lahijanian, M. Kloetzer, S. Itani, C. Belta, and S. Andersson, "Automatic deployment of autonomous cars in a robotic urban-like environment (RULE)," in *Proc. IEEE Int. Conf. Robot. Autom.*, Kobe, Japan, Oct. 2009, pp. 2055–2060.

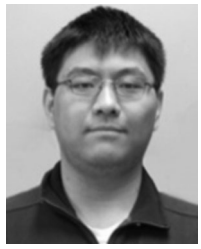
[32] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive(1) designs," in *Proc. Int. Conf. Verif., Model Check., Abstract Interpret.*, Charleston, SC, Jan. 2006, pp. 364–380.



Yushan Chen (S'10) received the B.S. degree in computer engineering from Beijing University of Post and Telecommunication, Beijing, China, in 2008. She is currently working toward the Ph.D. degree in electrical and computer engineering with Boston University, Boston, MA.

Her research interests include coordination and control of multiagent systems, robot motion planning, and formal methods in control synthesis.

Ms. Chen received the Best Student Paper Award at the International Symposium on Distributed Autonomous Robotic Systems in 2010.



Xu Chu (Dennis) Ding (M'10) received the B.S., M.S., and Ph.D. degrees in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, in 2004, 2008, and 2009, respectively.

He is currently a Postdoctoral Fellow with the Department of Mechanical Engineering, Boston University, Boston, MA. His research interests include formal methods in control synthesis, optimal control of hybrid systems, coordination and control of multi-agent networked systems, and intelligent and persistent surveillance.

Alin Stefanescu (M'11) received the M.Sc. degree from the University of Bucharest, Bucharest, Romania, in 2000, and the Ph.D. degree in computer science from the University of Stuttgart, Stuttgart, Germany, in 2006.

He is currently a Researcher with the University of Pitesti, Pitesti, Romania. His research career path has included both academic and industry. In academia, he investigated software validation, verification, and synthesis at European Universities in Edinburgh, U.K; Munich, Germany; Bucharest, Romania; and Konstanz, Germany. In industry, at SAP Research, Darmstadt, Germany, he made contributions to model-based testing for service-oriented architectures and transferring the results into the SAP development groups.

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, in 1995 and 1996, respectively, the M.Sc. degree in electrical engineering from Louisiana State University, Baton Rouge, in 1999, and the M.Sc. and Ph.D. degrees in mechanical engineering from the University of Pennsylvania, Philadelphia, in 2001 and 2003, respectively.

He is currently an Assistant Professor with the Boston University, Boston, MA. His research interests include the analysis and control of hybrid systems, motion planning and control, and biomolecular networks.

Dr. Belta is an Associate Editor for the *SIAM Journal on Control and Optimization*. He received the Air Force Office of Scientific Research Young Investigator Award in 2008 and the National Science Foundation CAREER Award in 2005.