2011 50th IEEE Conference on Decision and Control and
European Control Conference (CDC-ECC)
Orlando, FL, USA, December 12-15, 2011

Synthesis of Distributed Control and Communication Schemes from Global LTL Specifications

Yushan Chen, Xu Chu Ding, and Calin Belta

Abstract— **We introduce a technique for synthesis of control and communication strategies for a team of agents from a global task specification given as a Linear Temporal Logic (LTL) formula over a set of properties that can be satisfied by the agents. We consider a purely discrete scenario, in which the dynamics of each agent is modeled as a finite transition system. The proposed computational framework consists of two main steps. First, we extend results from concurrency theory to check whether the specification is distributable among the agents. Second, we generate individual control and communication strategies by using ideas from LTL model checking. We apply the method to automatically deploy a team of miniature cars in our Robotic Urban-Like Environment.**

I. Introduction

In control problems, "complex" models, such as systems of differential equations, are usually checked against "simple" specifications, such as the stability of an equilibrium, the invariance of a set, controllability, and observability. In formal synthesis (verification), "rich" specifications such as languages and formulas of temporal logics are checked against "simple" models of software programs and digital circuits, such as (finite) transition systems. Recent studies show promising possibilities to bridge this gap by developing theoretical frameworks and computational tools, which allow one to synthesize controllers for continuous and hybrid systems satisfying specifications in rich languages. Examples include Linear Temporal Logic (LTL) [1], fragments of LTL [2], Computation Tree Logic (CTL) [3], mu-calculus [4], and regular expressions [5].

A fundamental challenge in this area is to construct finite models that accurately capture behaviors of dynamical systems. Recent approaches are based on the notion of abstraction [6] and equivalence relations such as simulation and bisimulation [7]. Enabled by recent developments in hierarchical abstractions of dynamical systems [1], it is now possible to model systems with linear dynamics [8], polynomial dynamics [9], and nonholonomic (unicycle) dynamics [10] as finite transition systems.

More recent work suggests that such hierarchical abstraction techniques for a single agent can be extended to multi-agent systems, using parallel compositions [3], [11]. The two main limitations of this approach are the state space explosion problem and the need for frequent agent synchronization. In [5], we addressed both of these limitations with a "top-down" approach, by drawing inspirations from distributed formal synthesis [12]. The main idea is to decompose a global specification into local specifications, which can then be used to synthesize controllers for the individual agents. The top-down approach allows parallel execution among the agents, while guaranteeing that the global behavior of the team satisfies the global specification. This cannot be achieved by modeling the agents as a synchronous or asynchronous parallel composition and simply using a regular LTL model checker to generate a solution. The main drawback of the method in [5] is that the expressivity is limited to regular languages.

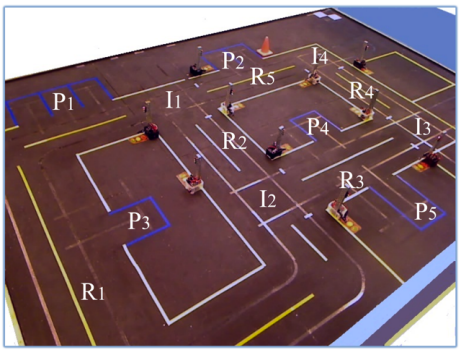

Fig. 1. The topology of the Robotic Urban-Like Environment (RULE) and the road, intersection, and parking lot labels.

In this paper, we address a purely discrete problem, in which each agent is modeled as a finite transition system: **Given** 1) a set of properties of interest that need to be satisfied, 2) a team of agents and their capacities and cooperation requirements for satisfying properties, 3) a task specification describing how the properties need to be satisfied subject to some temporal and logical constraints in the form of an LTL formula over the set of properties; **Find** provably-correct individual control and communication strategies for each agent such that the task is accomplished. Drawing inspiration from the areas of concurrency theory [13] and distributed formal synthesis [12], we develop a top-down approach that allows for the fully automatic synthesis of individual control and communication schemes. This framework is quite general and can be used in conjunction with abstraction techniques to control multiple agents with continuous dynamics.

The contribution of this work is threefold. First, we develop a computational framework to synthesize individual control and communication strategies from global specifications given as LTL formulas over a set of interesting

This work was partially supported by ONR MURI N00014-09-1051, ARO W911NF-09-1-0088, AFOSR YIP FA9550-09-1-020 and NSF CNS-0834260 at Boston University. Y. Chen is with the Department of Electrical and Computer Engineering, Boston University, Boston, USA, yushanc@bu.edu; X.C. Ding and C. Belta are with the Department of Mechanical Engineering, Boston University, Boston, USA, {xcding,cbelta}@bu.edu. Y. Chen is the corresponding author.

properties. This is a significant improvement over [5] by increasing the expressivity of specifications. Second, we extend the approach of checking closure properties of temporal logic specifications in [14] to generate distributed control and communication strategies for a team of agents while considering their dynamics. Specifically, we show how a satisfying distributed execution can be found when the global specification is traced-closed. Third, we implement and illustrate the computational framework in our Khepera-based Robotic Urban-Like Environment (RULE) (Fig. 1). In this experimental setup, robotic cars can be automatically deployed from specifications given as LTL formulas to service requests that occur at the different locations while avoiding the unsafe regions.

II. Preliminaries

For a set Σ, we use $|\Sigma|$, 2^{Σ}, Σ^*, and Σ^ω to denote its cardinality, power set, set of finite words, and set of infinite words, respectively. We define $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ and denote the empty word by ϵ.

*Definition 1 (**transition system**):* A transition system (TS) is a tuple $\mathcal{T} := (S, s_0, \rightarrow, \Sigma, h)$, consisting of (i) a finite set of states S; (ii) an initial states $s_0 \in S$; (iii) a transition relation $\rightarrow \subseteq S \times S$; (iv) a finite set of properties Σ; and (v) an output map $h : S \rightarrow \Sigma$.

A transition $(s, s') \in \rightarrow$ is also denoted by $s \rightarrow s'$. Properties can be either true or false at each state of $\mathcal{T}$. The output map $h(s)$, where $s \in S$, defines the property valid at state s. A finite (infinite) *trajectory* of $\mathcal{T}$ is a finite sequence $r_\mathcal{T} = s(0)s(1)\dots s(n)$ $(s(0)s(1)\dots)$ such that $s(0) = s_0$ and $s(i) \rightarrow s(i+1)$, $\forall i \geq 0$. A finite or infinite trajectory generates a finite or infinite *word* as a sequence of properties valid at each state, respectively.

We employ Linear Temporal Logic (LTL) formulas to express global tasks for a team of agents. Informally, LTL formulas are built from a set of properties Σ, standard Boolean operators $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), and temporal operators $\bigcirc$ (next), $\mathcal{U}$ (until), $\Diamond$ (eventually), $\Box$ (always). The semantics of LTL formulas are given over infinite words w over Σ, such as those generated by a transition system defined in Def. 1. We say an infinite trajectory $r_\mathcal{T}$ of $\mathcal{T}$ satisfies an LTL formula ϕ iff the word generated by $r_\mathcal{T}$ satisfies ϕ.

A word satisfies an LTL formula ϕ if ϕ is true at the first position of the word; $\bigcirc\phi$ states that at the next state, an LTL formula ϕ is true; $\Diamond\phi$ means that ϕ eventually becomes true in the word; $\Box\phi$ means that ϕ is true at all positions of the word; $\phi_1 \,\mathcal{U}\phi_2$ means ϕ_2 eventually becomes true and ϕ_1 is true until this happens. More expressivity can be achieved by combining the above temporal and Boolean operators, *e.g.,* $\Box\Diamond\phi$ (ϕ is true infinitely often) and $\Diamond\phi_1 \wedge \Box\neg\phi_2$ (ϕ_1 eventually becomes true and ϕ_2 is always avoided).

For every LTL formula ϕ, there exists a Büchi automaton accepting all and only the words satisfying ϕ [15]. We refer readers to [16] and references therein for efficient algorithms and freely downloadable implementations to translate a LTL formula ϕ to a corresponding Büchi automaton.

*Definition 2 (**Büchi automaton**):* A Büchi automaton (BA) is a tuple $\mathcal{B} := (Q, Q^{in}, \Sigma, \delta, F)$, consisting of (i) a finite set of states Q; (ii) a set of initial states $Q^{in} \subseteq Q$; (iii) an input alphabet Σ; (iv) a transition function $\delta : Q \times \Sigma \rightarrow 2^Q$; (v) a set of accepting states $F \subseteq Q$.

A *run* of the Büchi automaton over an infinite word $w = w(0)w(1)\dots$ over Σ is a sequence $r_\mathcal{B} = q(0)q(1)\dots$, such that $q(0) \in Q^{in}$ and $q(i+1) \in \delta(q(i), w(i))$. A Büchi automaton accepts a *word* w iff $\exists r_\mathcal{B}$ over w such that $\mathit{inf}(r_\mathcal{B}) \cap F \neq \emptyset$, where $\mathit{inf}(r_\mathcal{B})$ denotes the set of states appearing infinitely often in run $r_\mathcal{B}$. The *language* accepted by a Büchi automaton, denoted by $\mathcal{L}(\mathcal{B})$, is the set of all infinite words accepted by $\mathcal{B}$. We use $\mathcal{B}_\phi$ to denote the Büchi automaton accepting the language satisfying ϕ.

Remark 1: In LTL model checking [15], several properties can be valid at one state of a transition system (also called Kripke structure). The words produced by a transition system and accepted by a Büchi automaton are over the power set of propositions. In this paper, by allowing only one property to be valid at a state, we consider a particular case where we allow only one property to be valid at each state of a TS by defining h as a mapping from S to Σ. As a consequence, the words generated by $\mathcal{T}$ and accepted by $\mathcal{B}$ are over Σ.

*Definition 3 (**distribution**):* Given a set Σ, a collection of subsets $\{\Sigma_i \subseteq \Sigma, i \in I\}$ is called a distribution of Σ if $\cup_{i\in I}\Sigma_i = \Sigma$, where I is an index set.

*Definition 4 (**projection**):* For a word $w \in \Sigma^\infty$ and a subset $S \subseteq \Sigma$, we denote by $w \restriction_S$ the projection of w onto S, which is obtained by erasing all symbols σ in w that do not belong to Σ. For a language $L \subseteq \Sigma^\infty$ and a subset $S \subseteq \Sigma$, we denote by $L \restriction_S$ the projection of L onto S, which is given by $L \restriction_S := \{\omega \restriction_S \mid \omega \in L\}$.

*Definition 5 (**trace-closed language**):* Given a distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$ and $w, w' \in \Sigma^\infty$, we say that w is trace-equivalent to w' ($w \sim w'$ [1]) iff $w \restriction_{\Sigma_i} = w' \restriction_{\Sigma_i}$, $\forall i \in I$. We denote by $[w]$ the trace-equivalence class of $w \in \Sigma^\infty$, which is given by $[w] := \{w' \in \Sigma^\infty \mid w \sim w'\}$. A trace-closed language over a distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$ is a language L such that $\forall w \in L$, $[w] \subseteq L$.

*Definition 6 (**product of languages**):* Given a distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$, the product of a set of languages L_i over Σ_i is denoted by $\|_{i\in I}\, L_i$ and defined as $\|_{i\in I}\, L_i := \{w \in \Sigma^\infty \mid w \restriction_{\Sigma_i} \in L_i \text{ for all } i \in I\}$.

We refer to [13], [17] for more definitions and properties in concurrency theory.

III. Problem Formulation and Approach

Assume we have a team of agents $\{i \mid i \in I\}$, where I is a label set. We use an LTL formula over a set of properties Σ to describe a global task for the team. We model the capabilities of the agents to satisfy properties as a distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$, where Σ_i is the set of properties that can be satisfied by agent i. A property can be *shared* or *individual*, depending on whether it belongs to multiple agents or to a

[1]Note that the trace-equivalence relation $\sim$ and class $[\cdot]$ are based on the given distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$. For simplicity of notation, we use $\sim$ and $[\cdot]$ without specifying the distribution when there is no ambiguity.

single agent. Shared properties are properties that need to be satisfied by several agents simultaneously.

We model each agent as a transition system:

$$\mathcal{T}_i = (S_i, s_{0_i}, \rightarrow_i, \Sigma_i, h_i), i \in I. \quad (1)$$

In other words, the dynamics of agent i are restricted by the transition relation $\rightarrow_i$. The output $h_i(s_i)$ represents the property that is valid (true) at state $s_i \in S_i$. An individual property σ is said to be satisfied if and only if the agent that owns σ reaches state s_i at which σ is valid (*i.e.,* , $h_i(s_i) = \sigma$). A shared property is said to be satisfied if and only if all the agents sharing it enter the states where σ is true simultaneously.

For example, $\mathcal{T}_i$ can be used to model the motion capabilities of a robot (Khepera III miniature car) running in RULE (Fig. 1), where S_i is a set of labels for the roads, intersections and parking lots and $\rightarrow_i$ shows how these are connected (*i.e.,* $\rightarrow_i$ captures how robot i can move among adjacent regions). Note that these transitions are, in reality, enabled by low-level control primitives (see Sec. V). We assume that the selection of a control primitive at a region uniquely determines the next region. This corresponds to a deterministic (control) transition system, in which each trajectory of $\mathcal{T}_i$ can be implemented by the robot in the environment by using the sequence of corresponding motion primitives. For simplicity of notation, since the robot can deterministically choose a transition, we omit the control inputs traditionally associated with transitions. Furthermore, distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$ can be used to capture the capabilities of the robots to service requests and task cooperation requirements (*e.g.,* some of the requests can be serviced by one robot, while others require the collaboration of two or more robots). The output map h_i indicates the locations of the requests. A robot services a request by visiting the region at which this request occurs. A shared request occurring at a given location requires multiple robots to be at this location at the same time.

*Definition 7 (**cc-strategy**):* A finite (infinite) trajectory $r_i^c = s_i(0)s_i(1)\dots s_i(n)$ $(s_i(0)s_i(1)\dots)$ of $\mathcal{T}_i$ defines a control and communication (cc) strategy for agent i in the following sense: (i) $s_i(0) = s_{0_i}$, (ii) an entry $s_i(k)$ means that state $s_i(k)$ should be visited, (iii) an entry $s_i(k)$, where $h_i(s_i(k))$ is a shared property, triggers a communication protocol: while at state $s_i(k)$, agent i broadcasts the property $h_i(s_i(k))$ and listens for broadcasts of $h_i(s_i(k))$ from all other agents that share the property with it; when they are all received, $h_i(s_i(k))$ is satisfied and then agent i transits to the next state.

Because of the possible parallel satisfaction of individual properties, and because the durations of the transitions are not known, a set of cc-strategies can produce multiple sequences of properties satisfied by the team. We use products of languages to capture all the possible behaviors of the team.

*Definition 8 (**global behavior of the team**):* Given a set of cc-strategies $\{r_i^c, i \in I\}$, we denote $\mathcal{L}_{team}(\{r_i^c, i \in I\}) := \|_{i \in I} \{w_i\}$ as the set of all possible sequences of properties satisfied by the team while the agents follow their individual cc-strategies r_i^c, where w_i is the word of $\mathcal{T}_i$ generated by r_i^c.

For simplicity of notation, we usually denote $\mathcal{L}_{team}(\{r_i^c, i \in I\})$ as $\mathcal{L}_{team}$ when there is no ambiguity.

*Definition 9 (**satisfying set of cc-strategies**):* A set of cc-strategies $\{r_i^c, i \in I\}$ satisfies a specification given as an LTL formula ϕ iff $\mathcal{L}_{team} \neq \emptyset$ and $\mathcal{L}_{team} \subseteq \mathcal{L}(\mathcal{B}_\phi)$.

Remark 2: For a set of cc-strategies, the corresponding $\mathcal{L}_{team}$ could be an empty set by the definition of product of languages. In practice, this case corresponds to a deadlock scenario where one (or more) agent waits indefinitely for others to enter the states at which a shared property σ is true. For example, if one of these agents is not going to broadcast σ but some other agents are waiting for the broadcasts of σ, then all those agents will be stuck in a deadlock state and wait indefinitely. When such a deadlock scenario occurs, the behaviors of the team do not satisfy the specification.

We are now ready to formulate the main problem:

Problem 1: Given a team of agents represented by $\mathcal{T}_i$, $i \in I$, a global specification ϕ in the form of an LTL formula over Σ, and a distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$, find a satisfying set of individual cc-strategies $\{r_i^c, i \in I\}$.

Our approach can be divided into two major parts : checking distributability and ensuring implementability. Specifically, we (i) check whether the global specification can be distributed among the agents while accounting for their capabilities to satisfy properties, and (ii) make sure that the individual cc-strategies are feasible for the agents. For (i), we make the connection between distributability of global specifications and closure properties of temporal logic formulas [14]. Specifically, we check whether the language satisfying the global specification ϕ is trace-closed; if yes, then it is distributable; otherwise, our approach cannot find a solution, even though one may exist (Sec. IV-A). Therefore, our approach is conservative, in the sense that we might not find a solution even if one exists. For (ii), we construct an implementable automaton by adapting automata-based techniques [18], [19] to obtain all the possible sequences of properties that could be satisfied by the team, while considering the dynamics and capabilities of the agents (Sec. IV-B and IV-C). Finally, an arbitrary word from the intersection of the trace-closed language satisfying ϕ and the language of the implementable automaton is selected to synthesize the individual cc-strategies for the agents.

IV. Synthesis of Individual CC-Strategies

We omit all the proofs due to space limitations. They are available in [20].

A. Checking Distributability

We begin with the conversion of the global specification ϕ over Σ to a Büchi automaton (BA) $\mathcal{B}_\phi = (Q, Q^{in}, \Sigma, \delta, F)$, which accepts exactly the language satisfying ϕ (using LTL2BA [16]). We aim to find a local word w_i for each agent such that all possible sequences of properties satisfied by the team while each agent executes its local word satisfy the global specification ϕ.

Given ϕ and the distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$, we make the important observation that a trace-closed language (Def.

5) is sufficient to find a set of local words satisfying the condition. Formally, we have:

Proposition 1: Given a language $L \subset \Sigma^\infty$ and a distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$, if L is a trace-closed language, then $\forall w \in L$, we have $\|_i \{w \upharpoonright_{\Sigma_i}\} \subseteq L$.

Thus, our approach aims to check whether $\mathcal{L}(\mathcal{B}_\phi)$ is trace-closed. If the answer is positive, by Prop. 1, an arbitrary word from $\mathcal{L}(\mathcal{B}_\phi)$ can be used to generate the suitable set of local words by projecting this word onto Σ_i. The algorithm (adapted from [14]) to check if $\mathcal{L}(\mathcal{B}_\phi)$ is trace-closed can be viewed as a process to construct a BA $\mathcal{A}$, such that each word accepted by $\mathcal{A}$ represents a pair of words w and w', such that $w \in \mathcal{L}(\mathcal{B}_\phi)$, $w' \notin \mathcal{L}(\mathcal{B}_\phi)$, and $w \sim w'$ (*i.e.,* w is trace-equivalent to w'). Thus, if $\mathcal{A}$ has a non-empty language, $\mathcal{L}(\mathcal{B}_\phi)$ is not trace-closed (see [14]).

To obtain $\mathcal{A}$, we first construct a BA, denoted by $\mathcal{C}$, to capture all pairs of trace-equivalent infinite words over Σ. Given the distribution $\{\Sigma_i \subseteq \Sigma, i \in I\}$, we define a relation $\mathbb{I}$ such that $(\sigma, \sigma') \in \mathbb{I}$ if Σ_i, $i \in I$ such that $\sigma, \sigma' \in \Sigma_i$. Formally, $\mathcal{C}$ is defined as

$$\mathcal{C} = (Q_\mathcal{C}, \{q_{\mathcal{C}_0}\}, \Sigma_\mathcal{C}, \delta_\mathcal{C}, F_\mathcal{C}), \tag{2}$$

where $\Sigma_\mathcal{C} = \mathbb{I} \cup \{(\sigma, \sigma) \mid \sigma \in \Sigma\}$ and $F_\mathcal{C} = \{q_{\mathcal{C}_0}\}$. The transition function $\delta_\mathcal{C}$ is defined as 1) $\forall \sigma \in \Sigma$, $\exists q_{\mathcal{C}_0} = \delta_\mathcal{C}(q_{\mathcal{C}_0}, (\sigma, \sigma))$, and 2) $\forall (\sigma, \sigma') \in \mathbb{I}$, $\exists q_\mathcal{C} \neq q_{\mathcal{C}_0}$ such that $q_\mathcal{C} = \delta_\mathcal{C}(q_{\mathcal{C}_0}, (\sigma, \sigma'))$ and $q_{\mathcal{C}_0} = \delta_\mathcal{C}(q_\mathcal{C}, (\sigma', \sigma))$.

Next, we construct a BA $\mathcal{A}_1$ to accommodate words from $\mathcal{L}(\mathcal{B}_\phi)$. A word $w_{\mathcal{A}_1}$ accepted by $\mathcal{A}_1$ is a sequence $(\sigma_1, \sigma_1')(\sigma_2, \sigma_2')\ldots$. We use $w_{\mathcal{A}_1}|_1$ and $w_{\mathcal{A}_1}|_2$ to denote the sequence $\sigma_1\sigma_2\ldots$ and $\sigma_1'\sigma_2'\ldots$, respectively. For each word $w_{\mathcal{A}_1}$ accepted by $\mathcal{A}_1$, we have $w_{\mathcal{A}_1}|_1 \in \mathcal{L}(\mathcal{B}_\phi)$ and $w_{\mathcal{A}_1}|_2 \in \Sigma^\omega$. Similarly, we construct another BA $\mathcal{A}_2$ to capture words that do not belong to $\mathcal{L}(\mathcal{B}_\phi)$, *i.e.,* $\forall w_{\mathcal{A}_2} \in \mathcal{L}(\mathcal{A}_2)$, we have $w_{\mathcal{A}_2}|_1 \in \Sigma^\omega$ and $w_{\mathcal{A}_2}|_2 \notin \mathcal{L}(\mathcal{B}_\phi)$.

Finally, we produce $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{C}) \cap \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ by taking the intersections of $\mathcal{C}$, $\mathcal{A}_1$ and $\mathcal{A}_2$ (see [21]). $\mathcal{L}(\mathcal{B}_\phi)$ is trace-closed iff $\mathcal{L}(\mathcal{A}) = \emptyset$. We summarize this procedure in Alg. 1.

Algorithm 1 : Check if $\mathcal{L}(\mathcal{B})$ is trace-closed

Input: A BA $\mathcal{B} = (Q, Q^{in}, \Sigma, \delta, F)$ and $\{\Sigma_i \subseteq \Sigma, i \in I\}$
Output: Yes or No

1: Construct $\mathcal{C}$ as defined in (2)
2: Construct $\mathcal{A}_1 = (Q, Q^{in}, \Sigma_{\mathcal{A}_1}, \delta_{\mathcal{A}_1}, F)$, where $\Sigma_{\mathcal{A}_1} \subseteq \Sigma \times \Sigma$ and $\delta_{\mathcal{A}_1} : Q \times \Sigma_{\mathcal{A}_1} \to 2^Q$ is defined as $q' \in \delta_{\mathcal{A}_1}(q, (\sigma_1, \sigma_2))$ iff $q' \in \delta(q, \sigma_1)$
3: Construct $\mathcal{A}_2 = (Q, Q^{in}, \Sigma_{\mathcal{A}_2}, \delta_{\mathcal{A}_2}, F)$, where $\Sigma_{\mathcal{A}_2} \subseteq \Sigma \times \Sigma$ and $\delta_{\mathcal{A}_2} : Q \times \Sigma_{\mathcal{A}_2} \to 2^Q$ is defined as $q' \in \delta_{\mathcal{A}_2}(q, (\sigma_1, \sigma_2))$ iff $q' \in \delta(q, \sigma_2)$.
4: Construct $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{C}) \cap \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$
5: **if** $\mathcal{L}(\mathcal{A}) = \emptyset$ **return** Yes **else return** No

B. Implementable Local Specification

In the case that $\mathcal{L}(\mathcal{B}_\phi)$ is trace-closed, the global specification is distributable among the agents. We call $\mathcal{L}(\mathcal{B}_\phi) \upharpoonright_{\Sigma_i}$ the "local" specification for agent i because of Prop. 2.

Proposition 2: If a set of cc-strategies $\{r_i^c, i \in I\}$ is a solution to Prob. 1, then the corresponding local words w_i^c are included in $\mathcal{L}(\mathcal{B}_\phi) \upharpoonright_{\Sigma_i}$, $\forall i \in I$.

Given the agent model $\mathcal{T}_i$, some of the local words might not be feasible for the agent. Therefore, we aim to construct the "implementable local" specification for each agent; namely, it captures all the words of $\mathcal{L}(\mathcal{B}_\phi) \upharpoonright_{\Sigma_i}$ that can be implemented by the agent. To achieve this, we first produce an automaton that accepts exactly the local specification.

Note that the projection of the language satisfying the global specification that includes only infinite words on a local alphabet Σ_i might contain finite words. Therefore, the local specification for each agent might have both finite and infinite words. To address this, we employ a *mixed Büchi automaton* introduced in [19]. The mixed Büchi automaton is similar to the standard BA defined in Def. 2, except for it has two different types of accepting states: finitary and infinitary accepting states. Formally, we define the mixed Büchi automaton as

$$\mathcal{B}^M := (Q, Q^{in}, \Sigma, \delta, F, F^{fin}) \tag{3}$$

where F stands for the set of infinitary accepting states and F^{fin} represents the set of finitary accepting states. The mixed Büchi automaton accepts infinite words by using the set of infinitary accepting states, with the same acceptance condition as defined in Def. 2. A finite run $r^{fin} = q(0)q(1)\ldots q(n)$ of $\mathcal{B}^M$ over a finite word $w^{fin} = w(0)w(1)\ldots w(n)$ satisfies $q(0) \in Q^{in}$ and $q(i+1) \in \delta(q(i), w(i))$, $\forall 0 \leq i < n$. $\mathcal{B}^M$ accepts a finite word w^{fin} iff the finite run r^{fin} over w^{fin} satisfying $q(n) \in F^{fin}$. An algorithm to $\mathcal{B}_i$ accepting $\mathcal{L}(\mathcal{B}_\phi) \upharpoonright_{\Sigma_i}$ can be found in [20].

Inspired from LTL model checking [18], we define the following product automaton to obtain the implementable local specification. This automaton captures all the words in $\mathcal{L}(\mathcal{B}_i)$ that can be generated by agent i.

Definition 10: The product automaton $E_i = \mathcal{T}_i \otimes \mathcal{B}_i$ between a TS $\mathcal{T}_i = (S_i, s_{0_i}, \to_i, \Sigma, h_i)$ and a mixed BA $\mathcal{B}_i = (Q_i, Q_i^{in}, \Sigma_{\mathcal{B}_i}, \delta_i, F_i, F_i^{fin})$, is a mixed BA $E_i = (Q_{E_i}, Q_{E_i}^{in}, \Sigma_{E_i}, \delta_{E_i}, F_{E_i}, F_{E_i}^{fin})$ consisting of (i) a set of states $Q_{E_i} = S_i \times Q_i$; (ii) a set of initial states $Q_{E_i}^{in} = \{Start\} \times Q_i^{in}$; (iii) a set of inputs $\Sigma_{E_i} = \Sigma_{\mathcal{B}_i}$; (iv) a transition function δ_{E_i} defined as $(s_{0_i}, q') \in \delta_{E_i}((Start, q), h_i(s_{0_i}))$ iff $q' \in \delta_i(q, h_i(s_{0_i}))$ and $(s', q') \in \delta_{E_i}((s, q), h_i(s'))$ iff $s \to_i s'$ and $q' \in \delta_i(q, h_i(s'))$; (v) a set of infinitary accepting states $F_{E_i} = S_i \times F_i$; (vi) a set of finitary accepting states $F_{E_i}^{fin} = S_i \times F_i^{fin}$.

We add a dummy initial state $Start$ in order to capture the property satisfied at state s_{0_i}. We modify the traditional definition of product automata [15] to accommodate the finitary accepting states. Prop. 3 shows that $\mathcal{L}(E_i)$ is exactly the implementable local specification for agent i.

Proposition 3: Given any $w \in \mathcal{L}(\mathcal{B}_i)$, there exist at least one trajectory of $\mathcal{T}_i$ generating w iff $w \in \mathcal{L}(E_i)$.

C. Implementable Global Behaviors

To solve Prob. 1, we need to select a word w satisfying the global specification and also guarantee that $w_i = w \upharpoonright_{\Sigma_i}$

is executable for all the agents. Such a word can be obtained from the intersection of the global specification and the implementable global behaviors of the team, which can be modeled by the synchronous product of E_i.

Definition 11 ([19]): The synchronous product of n mixed Büchi automata $E_i = (Q_{E_i}, Q^{in}_{E_i}, \Sigma_{E_i}, \delta_{E_i}, F_{E_i}, F^{fin}_{E_i})$, denoted by $\|_{i=1}^{n} E_i$, is an automaton $\mathcal{P} = (Q_{\mathcal{P}}, Q^{in}_{\mathcal{P}}, \Sigma_{\mathcal{P}}, \delta_{\mathcal{P}})$, consisting of a set of states $Q_{\mathcal{P}} = Q_{E_1} \times \ldots \times Q_{E_n}$; a set of initial states $Q^{in}_{\mathcal{P}} = Q^{in}_{E_1} \times \ldots \times Q^{in}_{E_n}$; a set of inputs $\Sigma_{\mathcal{P}} = \cup_{i=1}^{n} \Sigma_{E_i}$; a transition function $\delta_{\mathcal{P}} : Q_{\mathcal{P}} \times \Sigma_{\mathcal{P}} \to 2^{Q_{\mathcal{P}}}$ defined as $q' \in \delta_{\mathcal{P}}(q, \sigma)$ such that if $i \in I_\sigma$, $q'[i] \in \delta_i(q[i], \sigma)$, otherwise $q[i] = q'[i]$, where $I_\sigma = \{i \in \{1, \ldots, n\} \mid \sigma \in \Sigma_i\}$ and $q[i]$ is the ith component of q.

The synchronous product composes n components, each of which represents the implementable local specification E_i for agent i. The synchronous product captures the synchronization among the agents as well as their parallel executions. Informally, a word w is accepted by $\mathcal{P}$ if and only if for each $i \in I$, $w \restriction_{\Sigma_i}$ is accepted by the corresponding component E_i. A method to find an accepted word of $\mathcal{P}$ is given in [19]. Note that $\mathcal{L}(\mathcal{P})$ is equal to the product of the languages of E_i (*i.e.,* $\|_{i \in I} \mathcal{L}(E_i)$). In other words, $\mathcal{L}(\mathcal{P})$ captures all possible global words that can be implemented by the team.

Finally, we can produce the solution to Prob. 1 by selecting an arbitrary word w from $\mathcal{L}(\mathcal{P}) \cap \mathcal{L}(\mathcal{B}_\phi)$, obtaining the local word $w_i = w \restriction_{\Sigma_i}$ and generating the corresponding cc-strategy r^c_i for each agent. The overall approach is summarized in Alg. 2. The following theorem shows that the output of Alg. 2 is indeed the solution to Prob. 1.

Algorithm 2 : Cc-strategies synthesis

Input: ϕ over Σ, $\{\Sigma_i \subseteq \Sigma, i \in I\}$, and $\{\mathcal{T}_i, i \in I\}$
Output: A set of cc-strategies $\{r^c_i, i \in I\}$

```
Convert φ to a Büchi automaton B_φ using LTL2BA [16]
if L(B_φ) is trace-closed (Alg. 1) then
    Construct B_i , E_i = T_i ⊗ B_i, P, and then an automaton
    accepting L(P) ∩ L(B_φ)
    if L(P) ∩ L(B_φ) ≠ ∅ then
        Obtain w ∈ L(P) ∩ L(B_φ), a set of local words
        {w_i = w ↾_{Σ_i}, i ∈ I}, and then construct a set of
        automata {B^c_i, i ∈ I}, where L(B^c_i) = w_i.
        Construct E^c_i = B^c_i ⊗ T_i, find an accepted run r_i of
        E^c_i, and project r_i on T_i to obtain r^c_i, ∀i ∈ I.
        return {r^c_i, i ∈ I}
    end if
end if
return solution not found
```

Theorem 1: If $\mathcal{L}(\mathcal{B}_\phi)$ is trace-closed, the set of cc-strategies $\{r^c_i, \in I\}$ obtained by Alg. 2 satisfies $\|_{i \in I} \{w_i\} \neq \emptyset$ and $\|_{i \in I} \{w_i\} \subseteq \mathcal{L}(\mathcal{B}_\phi)$, where w_i is the corresponding word of $\mathcal{T}_i$ generated by r^c_i.

Remark 3 (Completeness): In the case that $\mathcal{L}(\mathcal{B}_\phi)$ is trace-closed, our approach is complete in the sense that we find a solution to Prob. 1 if one exists. This follows directly from Prop. 3 and the fact that $\mathcal{L}(\mathcal{P}) = \|_{i \in I} \mathcal{L}(E_i)$. If $\mathcal{L}(\mathcal{B}_\phi)$ is not trace-closed, a complete solution to Prob. 1 requires one to find a non-empty trace-closed subset of $\mathcal{L}(\mathcal{B}_\phi)$ if one exists. This problem is not considered in this paper. Therefore, our overall approach to Prob. 1 is not complete.

Remark 4 (Complexity): From a computational complexity point of view, the bottlenecks of the presented approach are the computations relating to $\mathcal{P}$, because $|Q_{\mathcal{P}}|$ is bounded above by $\prod_{i \in I} |Q_{E_i}|$ and the upper bound of $|Q_{E_i}|$ is $O(|Q| \cdot |S_i|)$. For most robotic applications, the size of the task specification (*i.e.,* $|Q|$) is usually much smaller comparing to the size of the agent model (*i.e.,* $|S_i|$). Therefore, if we can shrink the size of Q_{E_i} by removing the information about the agent model from E_i, we can reduce the complexity significantly. Such reduction can be achieved by using LTL without the next operator and taking a stutter closure of E_i. This will be addressed in our future work.

V. Automatic Deployment in RULE

In this section, we show how our method can be used to deploy a team of robots (Khepera III) in our Robotic Urban-Like Environment (Fig. 1). The platform consists of a collection of roads, intersections, and parking lots. Each intersection has traffic lights. All the robots can communicate through Wi-Fi with a desktop computer, which is used as an interface to the user and to perform all the computation necessary to generate the individual cc-strategies. Once computed, these are sent to the robots, which execute the task autonomously by interacting with the environment and by communicating with each other, if necessary. We assume that inter-robot communication is always possible.

We model the motion of each robot in the platform using a transition system, as shown in Fig. 2. The set of states S_i is the set of labels assigned to roads, intersections and parking lots (see Fig. 1) and the relation $\to_i$ shows how these are connected. We distinguish one bound of a road from the other since the parking lots can only be located on one side of each road. For example, we use R_{1r} and R_{1l} to denote the two bounds of road R_1. Each state of $\mathcal{T}_i$ is associated with a set of motion primitives. For example, at region R_{1r}, which corresponds to the access point for parking lot P_1 (see Fig. 2), the robot can choose between two motion primitives: `follow_road` and `park`, which allow the robot to stay on the road or turn right into P_1. If the robot follows the road, it reaches the vertex I_2, where three motion primitives are available: `U_turn`, `turn_right_int` and `go_straight_int`, which allow the robot to make a U-turn, turn right or go straight through the intersection. It can be seen that, by selecting a motion primitive available at a region, the robot can correctly execute a trajectory of $\mathcal{T}_i$, given that it is initialized at a vertex of $\mathcal{T}_i$. The choice of a motion primitive uniquely determines the next vertex. In other words, a set of cc-strategies defined in Sec. III and obtained as described in Sec. IV can be immediately implemented by the team.

Assume that service requests, denoted by H_1, H_2, L_1, L_2 and L_3, occur at parking lots P_1, P_2, P_4, P_5 and P_3, respectively. "H" stands for "heavy" requests requiring the efforts

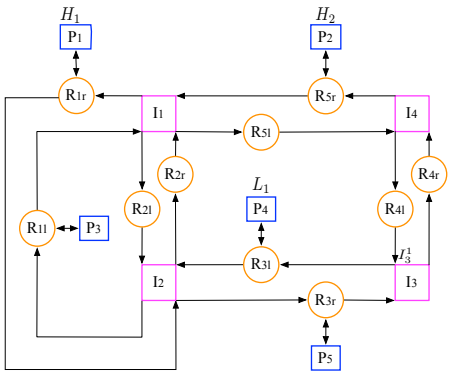

Fig. 2. Transition system $\mathcal{T}_1$ for robot 1 in RULE (Fig. 1). The output h_1 captures the locations of the unsafe regions and the requests. The dummy request ϖ_1 and the initial state are omitted in this figure.

of multiple robots while "L" represents "light" requests that only need one robot. We also consider some regions to be unsafe. In this example, we assume that intersection I_3 is unsafe for all robots before H_1 is serviced. We use the output map h_i of $\mathcal{T}_i$ to capture the locations of requests and unsafe regions. A "dummy request" ϖ_i is assigned to all the other regions. We use a special semantics for ϖ_i: a robot does not service any request when visiting a region where ϖ_i occurs.

We model the capabilities of the robots to service requests while considering unsafe regions as a distribution: $\Sigma_1 = \{H_1, H_2, L_1, I_3^1, \varpi_1\}, \Sigma_2 = \{H_1, H_2, L_2, I_3^2, \varpi_2\}$ and $\Sigma_3 = \{H_1, L_3, I_3^3, \varpi_3\}$. Note that we treat the unsafe region I_3 as an independent property assigned to each robot since it does not require the cooperation of the robots. The global specification ϕ is the conjunction of the following LTL formulas: 1) Request H_2 is serviced infinitely often: $\square\lozenge H_2$; 2) First service request H_1, then service request L_1 and L_2 regardless of the order or request L_3: $\lozenge(H_1 \wedge (\lozenge(L_1 \wedge L_2) \vee \lozenge L_3))$; 3) Do not visit intersection I_3 until H_1 is serviced: $\neg(I_3^1 \vee I_3^2 \vee I_3^3)\ \mathcal{U}\ H_1$.

By applying Alg. 2, we first learn that the language satisfying ϕ is trace-closed. Then, we obtain the implementable automaton $\|_{i\in I}\ E_i$ as described in Sec. IV-B and IV-C. Finally, we choose a word $w \in \mathcal{L}(\mathcal{B}_\phi) \cap \mathcal{L}(\|_{i\in I}\ E_i)$ and project w on the local alphabets Σ_i, $i \in \{1, 2, 3\}$ to obtain the local words, which lead to the following cc-strategies:

$$\begin{aligned} r_1^c &= R_{1r}I_2R_{2r}I_1R_{1r}P_1R_{1r}I_2R_{3r}I_3R_{4r}I_4R_{5r}P_2P_2\ldots, \\ r_2^c &= R_{5r}I_4R_{1r}P_1R_{1r}I_2R_{2r}I_1R_{5l}I_4R_{5r}P_2P_2\ldots, \\ r_3^c &= R_{2r}I_1R_{1r}P_1R_{1r}I_2R_{1l}P_3. \end{aligned}$$

The synchronization is only triggered when the robots are about to service shared requests, *i.e.,* when at P_1 and P_2. Besides these synchronization moments, the robots follow their cc-strategies and execute their individual tasks in parallel, which speed up the process of accomplishing the global task. The movie of the deployment in the RULE platform is available at `http://hyness.bu.edu/CDC2011`.

VI. Conclusions and Future Works

We present an algorithmic framework to deploy a team of agents from a task specification given as an LTL formula over a set of properties. Given the agent capabilities to satisfy the properties, and the possible cooperation requirements for the shared properties, we find individual control and communication strategies such that the global behavior of the system satisfies the given specification. We illustrate the proposed method with experimental results in RULE. For future work we are looking at ways to reduce the computational complexity. Another interesting direction is the extension to more realistic models of agents that can capture uncertainty and noise in the system, such as Markov Decision Processes.

References

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

[2] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas, "Courteous cars," *IEEE Robotics and Automation Magazine*, vol. 15, no. 1, pp. 30–38, 2008.

[3] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-robot motion planning: A timed automata approach," in *Proc ICRA*, Barcelona, Spain, April 2004, pp. 4417–4422.

[4] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc CDC*, Shanghai, China, 2009, pp. 2222 –2229.

[5] Y. Chen, X. Ding, A. Stefanescu, and C. Belta, "A formal approach to deployment of robotic teams in an urban-like environment," in *DARS*, Lausanne, Switzerland, November 2010.

[6] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, pp. 971–984, 2000.

[7] R. Milner, *Communication and Concurrency*. Prentice-Hall, 1989.

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

[9] A. Tiwari and G. Khanna, "Series of abstractions for hybrid automata," in *Proc HSCC*, ser. LNCS, vol. 2289, 2002, pp. 465–478.

[10] S. Lindemann, I. Hussein, and S. LaValle, "Real time feedback control for nonholonomic mobile robots with obstacles," in *Proc CDC*, New Orleans, LA, December 2007, pp. 2406–2411.

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

[12] M. Mukund, *From Global Specifications to Distributed Implementations*. Kluwer Academic Publishers, 2002, pp. 19–34.

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

[14] D. Peled, T. Wilke, and P. Wolper, "An algorithmic approach for checking closure properties of temporal logic specifications and omega-regular languages," *Theoretical Computer Science*, vol. 195, no. 2, pp. 183–203, 1998.

[15] C. Baier and J. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[16] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Proceedings of the Conference on Computer Aided Verification*, ser. LNCS, H. C. G. Berry and A. Finkel, Eds., no. 2102. Springer, 2001, pp. 53–65.

[17] M. Z. Kwiatkowska, "Event fairness and non-interleaving concurrency," in *Formal Aspects of Computing*, vol. 1, 1989, pp. 213–228.

[18] E. M. Clarke, D. Peled, and O. Grumberg, *Model Checking*. MIT Press, 1999.

[19] P. Thiagarajan, "PTL over product state spaces," Technical Report TCS-95-4, School of Mathematics, SPIC Science Foundation, Tech. Rep., 1995.

[20] Y. Chen, X. C. Ding, and C. Belta, "Synthesis of distributed control and communication schemes from global LTL specifications," Boston University, CISE Technical Report, Tracking Number 2011-IR-0004, (available at http://www.bu.edu/systems/publications/search-publications-database/).

[21] M. Y. Vardi and P. Wolper, "Reasoning about infinite computations," *Information and Computation*, vol. 115, pp. 1–37, 1994.