Synthesis for Multi-Robot Controllers with Interleaved Motion

Vasumathi Raman[1] and Hadas Kress-Gazit[2]

Abstract— **This paper addresses the problem of designing control schemes for teams of robots engaged in complex high-level tasks. It presents a method for automatically creating hybrid controllers that ensure that a team of possibly heterogeneous robots satisfies a user-defined high-level task. The proposed approach relaxes constraints on the simultaneous and interleaved motion of the robots, while maintaining constraints on their relative location to guarantee collision-avoidance and prevent deadlock. The approach is demonstrated in the context of a team of robots engaged in sorting objects for recycling.**

I. Introduction

Constructing controllers for high-level robot behaviors is an active field of research in robotics. The use of formal methods has seen much success in the construction of controllers for autonomous behaviors such as search and rescue missions and autonomous vehicle control [4], [8], [10], [12], [16]. With constantly improving technology for multi-robot applications such as robot swarms and self-driving cars, there is a growing need for system designers to be able to easily create such controllers for teams of robots.

Unlike approaches such as [9], [10] which consider multi-robot control for path planning problems, this work is concerned with reactive behaviors, which may require the system to behave differently based on gathered sensory information. Applications that involve such behaviors include search-and-rescue and autonomous driving. Non-reactive tasks, which include robots driving in formation to pre-specified goal positions, allow for hard-coded high-level behaviors with low-level tuning during execution. On the other hand, reactive behaviors require planning for every eventuality, adding complexity to an already nontrivial problem, and hand-coded controllers are infeasible for even modest problem sizes. The authors in [5] describe the automatic synthesis of control and communication strategies for a robotic team, from reactive task specifications of servicing requests in an environment. In contrast with the decentralized controllers presented in that work, this paper focuses on centralized controllers for a larger class of multi-robot tasks. Moreover, the authors in [5] assume that the location of the requests in the environment are known; in contrast, this work assumes an adversarial environment. Closest to the setting in this work, the authors in [13] design centralized controllers that satisfy reactive, temporal logic specifications, by composing controllers using navigation functions. However, the "secondary controllers" (i.e. controllers other than motion) used in the task definition are required to be continuous; the work presented in this paper does not restrict the non-motion controllers in this way.

In order to produce provably correct high-level plans from reactive specifications, techniques based on temporal logic synthesis have been applied to automatically synthesize correct-by-construction controllers from formal task specifications [12], [16]. These approaches operate on a discrete abstraction of the robot workspace, and a formal specification of the environment assumptions and desired robot behavior. The result is an automaton fulfilling the specification on this abstraction, if one exists. This automaton is in turn used to construct a hybrid controller that calls low-level continuous controllers to execute each discrete transition.

The challenge of constructing these continuous controllers for multi-robot systems was studied by [11]. The authors provided a technique for constructing atomic controllers that guarantee collision-avoidance and deadlock prevention for teams of robots. This work combined the production of high-level control in [12] with the low-level controller synthesis described in [2]. The atomic controllers described guide multiple robots to a goal set while avoiding collisions with obstacles and other robots, and can be reused to accommodate many different high-level tasks in the same workspace. In contrast with prior work, which achieved collision avoidance and deadlock prevention by imposing explicit user-defined constraints, the approach in [11] allows these constraints to be automatically generated during the construction of multi-robot atomic controllers, and admits a wider range of constraints, including minimum pairwise distances between robots.

Due to the nature of the multi-robot atomic controllers described in [11], the automaton representing the high-level plan was restricted to transitions in which only one robot moves at a time. This constraint is not only unnecessary for most tasks, but sometimes excessively restrictive, as demonstrated in Section IV. A key insight for overcoming this restriction is that, since motion is non-instantaneous, it can be abstracted to separate initiation and completion events. While several robot motion controllers may be initiated at once, the atomic controllers generated ensure that only one of them will *complete* over a single transition. This idea was first explored in the context of timing semantics for single-robot controllers with low-level actions of arbitrary relative execution durations [15]; its application to the multi-robot domain is described formally in this paper.

Vasumathi Raman is supported by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. Hadas Kress-Gazit is supported by NSF CAREER CNS-0953365 and NSF ExCAPE.

[1]V. Raman is with the Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125, USA vasu@caltech.edu.

[2]H. Kress-Gazit is with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853, USA hadaskg@cornell.edu

The rest of the paper is structured as follows. Section II defines the problem. Section III describes the construction of a multi-robot hybrid controller that provably accomplishes a high-level task. Section IV presents simulations, and shows how the approach differs from previous work. The paper concludes with a discussion in Section V.

II. Problem Formulation

This work considers a team of robots $A = \{a_i \mid i \in \{1,...,n\}\}$ moving in a polygonal workspace $W \subset \mathbb{R}^{d_i}$. Robot a_i has configuration $x_i \in X_i \subset \mathbb{R}^{d_i}$, and dynamics $\dot{x}_i = u_i$ for $u_i \in U_i$. The robots have minimum pairwise proximity constraints to ensure collision-avoidance.

Each robot a_i has a set of sensors $Sen = \{s_{i_j} \mid i = 1,...,n; j = 1,...,m_i\}$ that represent high-level information about the environment (i.e. discrete events that are not controlled by the robot, but relevant to the task at hand). This includes, for example, whether one of the robots is sensing a fire. Robots may also have a set of actions, $Act = \{act_{i_k} \mid i = 1,...,n; k = 1,...,l_i\}$, which correspond to discrete actions like picking up objects or transmitting messages. For this work, we assume that such actions do no have any timing constraints (i.e. they can be treated as instantaneous); the approach generalizes to actions with timing constraints.

Let $s_{i_j}(t)$ denote the value of sensor variable s_{i_j} at time t, and $act_{i_k}(t)$ denote the activation of actuator act_{i_j} at time t. Let $Sen_t = \{s_{i_j}(t) \mid s_{i_j} \in Sen\}$ denote the sensor values at time step t. A *strategy* Π for the robot system is a function such that $\Pi(Sen_t) = (u, Act_t)$, where $u = (u_1, u_2, ..., u_n)$ is a control vector and $Act_t = \{act_{i_k}(t) \mid act_{i_k} \in Act\}$ gives values to the robot actions. Let $Sen_\infty^\Pi = \{Sen_0, Sen_1, Sen_2, ...\}$, $x_\infty^\Pi = \{x_0, x_1, x_2, ...\}$, and $Act_\infty^\Pi = \{Act_0, Act_1, Act_2, ...\}$ denote the infinite sequences of sensor inputs, robot configurations and actions generated by following Π.

Finally, let φ be a high-level task specification provided in some formal problem description language. This specification describes both the desired behavior of the robots, and assumptions on the environment.

Problem 1 *Consider a team of robots A with the above dynamics and proximity constraints, moving on $\mathbb{R}^d$ where $d = \sum_{i=1}^n d_i$, with sensors Sen, actions Act, and high-level specification φ. For any possible initial state $\{x_0, Sen_0, Act_0\}$ such that $\{x_0, Sen_0, Act_0\} \models \varphi$, find a strategy Π, if one exists, such that:*

- $\dot{x}_i(t) = u_i(t)$
- *if $Sen_\infty^\Pi \models \varphi$ (i.e. the sensors satisfy the assumptions), then $(x_\infty^\Pi, Act_\infty^\Pi) \models \varphi$ (the robots fulfill the task)*

III. Controller Synthesis

This section presents the method used to construct a hybrid controller guaranteed to produce the specified multi-robot high-level behavior. Applying formal methods tools to the construction of provably correct multi-robot controllers involves (a) a discrete abstraction of the problem, in which the continuous reactive behavior of the multi-robot system is described in terms of a finite set of states, and (b) a temporal logic formalism for the specification, which in this work is Linear Temporal Logic (LTL) [6]. These components are now described, using an example to demonstrate the stages of synthesis for a simple high-level multi-robot recycling task.

A. Linear Temporal Logic (LTL)

Syntax: Let AP be a set of atomic propositions. LTL formulas are defined by the recursive grammar:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \; \mathcal{U} \; \varphi,$$

where $\pi \in AP$, $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is "next", and $\mathcal{U}$ is a strong "until". Conjunction ($\wedge$), implication ($\Rightarrow$), equivalence ($\Leftrightarrow$), "eventually" ($\Diamond$) and "always" ($\Box$) are derived from these operators.

Semantics: The truth of an LTL formula is evaluated over infinite sequences of states, corresponding to executions of a finite state machine representing the system. A state corresponds to an assignment of truth values to all propositions $\pi \in AP$. Given an infinite sequence of states σ, the statement $\sigma \models \varphi$ denotes that σ satisfies formula φ. The statement $\sigma \models \Delta\varphi$ for $\Delta = (\bigcirc, \Box, \Diamond, \Box\Diamond)$ denotes that φ is true at the second position, at every position, at some position, and infinitely often in σ, respectively. Let A be a finite state machine whose states are labeled with truth assignments on AP. Then A is said to satisfy φ if, for every execution σ of A, $\sigma \models \varphi$. The reader is referred to [6] for a formal definition of the semantics.

B. Discrete Abstraction

The relevant features of the continuous robot control problem are abstracted using a finite set of Boolean propositions. To capture the motion of the robots using the discrete formalism of LTL, the workspace is partitioned into regions, and Boolean variables introduced to indicate the location of each robot. To account for the non-instantaneous execution of continuous motion controllers, each motion is viewed as the *activation* of the corresponding low level controller, and a new sensor proposition is introduced in the discrete model to indicate whether the controller has completed execution. That is, the robot is able to sense when a motion controller has completed its action (i.e.,when it has crossed the boundary into a neighboring region).

The discrete abstraction consists of:

- $\pi_{s_{i_j}}$ for every sensor input s_{i_j} (e.g., $\pi_{1_{person}}$ is true if and only if robot 1 is sensing a person)
- $\pi_{act_{i_k}}$ for every robot action act_{i_k} (e.g., $\pi_{2_{camera}}$ is true if and only if the robot's camera is on)
- π_{i_r} for the *initiation* of motion by robot i towards region r (e.g., $\pi_{1_{office}}$ is true if and only if robot 1 is trying to move to the office). These propositions represent controller activation events.
- $\pi_{i_r}^c$ for the *completion* of robot i's motion to region r (e.g., $\pi_{1_{office}}^c$ is true if and only if robot 1 has arrived in the office). These propositions represent sensed events.

The set of sensor propositions is denoted by $\mathcal{X}$, and the set of action and location (i.e., robot-controlled) propositions by

$\mathcal{Y}$. Thus, for every sensor input s_{i_j}, robot action act_{i_k}, robot index i and region r, $\pi^c_{i_r}, \pi_{s_{i_j}} \in \mathcal{X}$ and $\pi_{i_r}, \pi_{act_{i_k}} \in \mathcal{Y}$. The value of each $\pi \in \mathcal{X} \cup \mathcal{Y}$ is the abstracted binary state of a low-level black box component, such as a thresholded sensor value or robot location with respect to some workspace partition.

Example 1 *Consider the four-room workspace depicted in Fig. 1(a), where the rooms are labeled r_1-r_4. There are two robots, indicated by circles: robot 1 starts in r_1 and robot 2 starts in r_4. Each robot has two sensors, one for glass and one for metal. Robot 1's specification is as follows. If it sees glass, it takes it to r_4, if it sees metal, it carries it to r_2. After it has done so, it proceeds to r_4 and stays there. Robot 2's specification is to go to r_1 and stay there. Here,*

1) $\mathcal{X} = \{\pi_{1_{glass}}, \pi_{2_{glass}}, \pi_{1_{metal}}, \pi_{2_{metal}}\} \cup \{\pi^c_{i_r} \mid i \in \{1,2\}, r \in \{1,2,3,4\}\}$
2) $\mathcal{Y} = \{\pi_{1_{carrying\_glass}}, \pi_{2_{carrying\_glass}}, \pi_{1_{carrying\_metal}}, \pi_{2_{carrying\_metal}}\} \cup \{\pi_{i_r} \mid i \in \{1,2\}, r \in \{1,2,3,4\}\}$

C. Task Specification

Given a discrete abstraction, a high-level task is specified using an LTL formula over $\mathcal{X} \cup \mathcal{Y}$. The task specification governs which actions can be activated by the robots, and assumptions on how the action-completion and environment sensors behave. There are two types of properties allowed in a specification: *safety* properties, which guarantee that "something bad never happens," and *liveness* conditions, which state that "something good (always eventually) happens".

This work considers tasks that can be specified using formulas of the form $\varphi_e \Rightarrow \varphi_s$, where φ_s represents the desired multi-robot system behavior, and φ_e encodes assumptions on the environment. The environment in this work includes events external to the robots as well as the system's internal state, as perceived by the robots' sensors. φ_e and φ_s each contain initial conditions $(\varphi^i_e, \varphi^i_s)$, safety requirements $(\varphi^t_e, \varphi^t_s)$ and liveness conditions $(\varphi^g_e, \varphi^g_s)$ for the environment and system, respectively. The following is an excerpt of the user-defined LTL specification for the task in Example 1:

$(\varphi^c_{1_{r_1}} \wedge \neg\pi_{1_{carrying\_metal}} \wedge \neg\pi_{1_{carrying\_glass}})$ #Initial
(Robot 1 starts in region r1, carrying nothing.)

$\wedge$ $(\varphi^c_{2_{r_4}} \wedge \neg\pi_{2_{carrying\_metal}} \wedge \neg\pi_{2_{carrying\_glass}})$ #Initial
(Robot 2 starts in region r4, carrying nothing.)

$\wedge$ $\square(\bigcirc \pi_{i_{glass}} \Rightarrow \bigcirc \pi_{i_{carrying\_glass}})$ #Safety
(Robot i activates carrying_glass if sensing glass)

$\wedge$ $\square(\bigcirc \pi_{i_{metal}} \Rightarrow \bigcirc \pi_{i_{carrying\_metal}})$ #Safety
(Robot i activates carrying_metal if sensing metal)

$\wedge$ $\square\diamondsuit(\pi_{1_{carrying\_glass}} \rightarrow \bigcirc \pi_{1_{r_4}})$ #Liveness
(If Robot 1 is carrying glass, it should visit r_4)

$\wedge$ $\square\diamondsuit(\pi_{1_{carrying\_metal}} \rightarrow \bigcirc \pi_{1_{r_2}})$ #Liveness
(If Robot 1 is carrying metal, it should visit r_2)

$\wedge$ $\square\diamondsuit(\pi_{1_{r_4}})$ #Liveness
(Robot 1 should go to r_4)

$\wedge$ $\square\diamondsuit(\pi_{2_{r_1}})$ #Liveness
(Robot 2 should go to r_1)

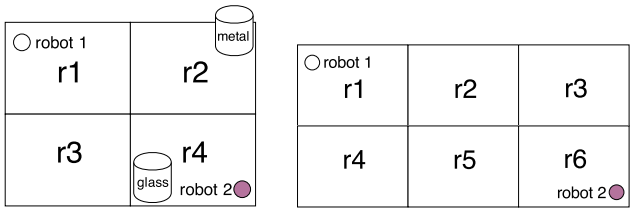
In addition to the user-defined specification, there are several constraints automatically generated depending on the availability of atomic motion controllers. These components of the specification are now described in detail.

1) Multi-Robot Motion Constraints: The robot's motion in the workspace is governed by the availability of controllers to drive it between adjacent regions, based on the location of the other robots and obstacles in the workspace. The atomic controllers used to drive the team of robots from one configuration to another can be constructed using a method such as the one presented in [11]. The construction of these atomic controllers produces an adjacency relation and safety conditions based on the workspace and the specified proximity constraints, which are automatically encoded as a logic formula φ_{motion}, and included as part of the specification.

In the construction of atomic controllers in [11], the configuration space is composed of polytopes in which the agents cannot collide, and each polytope is associated with a region combination based on where each robot is. The atomic controllers are generated by solving for linear feedback controllers that drive the team of robots from any state in one polytope to the exit facet shared with each adjacent polytope (i.e. each polytope that shares a matching facet). This results in paths between region combinations. Since the atomic controllers direct states to a facet, at most one robot will cross a room threshold at any time. In the approach presented in [11], this required restricting the possible motions of the team of robots in the specification (and thereby in the synthesized automaton), to those in which at most one robot changes rooms at a time. However, using the approach in this paper, the automaton no longer has to be restricted in this manner, and multiple robots may initiate motion simultaneously.

The initiation and completion of the linear feedback controllers for each robot are modeled as separate events, and therefore constraints on which robots can move (i.e. only one robot can move at a time) are replaced by assumptions on how these controllers can complete (at most one motion controller will complete at a time). This allows the activation of multiple robot motion controllers when possible, such as when the robots are operating in distant parts of the workspace. On the polytope graph, this corresponds to allowing paths between all polytopes in which each robot is in either the beginning or end configuration. Note that a single atomic controller corresponds to the activation or deactivation of several variables in the discrete abstraction (one for each robot). So an atomic controller that drives robot 1 from r_1 to r_2 and robot 2 from r_4 to r_3 will give rise to constraints on sensors corresponding to $\pi^c_{1_{r_1}}$ and $\pi^c_{2_{r_4}}$ and actions corresponding to $\pi_{1_{r_2}}$ and $\pi_{2_{r_3}}$.

The allowed motion of the robots depends on the sensed location and the availability of atomic controllers to drive the robots from one location to another. Since each robot can be in exactly one location at any given time, the formulas $\varphi^c_{i_r} = \pi^c_{i_r} \wedge \bigwedge_{r' \neq r} \neg\pi^c_{i_{r'}}$ and $\varphi_{i_r} = \pi_{i_r} \wedge \bigwedge_{r' \neq r} \neg\pi_{i_{r'}}$ are added to the specification to represent robot i being in and activating a controller to move to region r, respectively. Assume that for the above example, given the proximity constraints on



(a) Workspace for Example 1 (b) Workspace for Example 2

Fig. 1

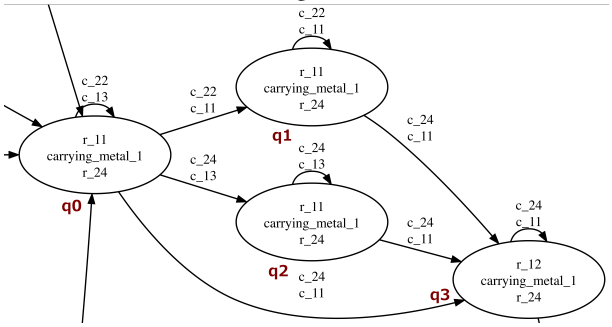


Fig. 2: Excerpt of synthesized automaton for Example 1

the robots and the availability of atomic controllers, they cannot be in the same region at the same time, but all other possible transitions are based on the adjacency of rooms. In other words, given initial and final configurations of the team of robots, there exists an atomic controller to drive from the initial configuration to the final configuration if and only if each robot is moving between adjacent rooms or staying in place, and the robots are not in the same room as each other in either configuration. Then the multi-robot motion polytope graph generated during the construction of the atomic controllers results in the following (automatically added) safety conditions in the specification for Example 1:

$$
\begin{aligned}
\varphi_{motion} = \;& \text{\# adjacency for robot } i \in \{1,2\} \\
& \Box(\varphi^c_{i_{r_1}} \Rightarrow \bigcirc \varphi_{i_{r_1}} \vee \bigcirc \varphi_{i_{r_2}} \vee \bigcirc \varphi_{i_{r_3}}) \\
& \wedge \Box(\varphi^c_{i_{r_2}} \Rightarrow \bigcirc \varphi_{i_{r_1}} \vee \bigcirc \varphi_{i_{r_2}} \vee \bigcirc \varphi_{i_{r_4}}) \\
& \wedge \Box(\varphi^c_{i_{r_3}} \Rightarrow \bigcirc \varphi_{i_{r_1}} \vee \bigcirc \varphi_{i_{r_3}} \vee \bigcirc \varphi_{i_{r_3}}) \\
& \wedge \Box(\varphi^c_{i_{r_4}} \Rightarrow \bigcirc \varphi_{i_{r_2}} \vee \bigcirc \varphi_{i_{r_3}} \vee \bigcirc \varphi_{i_{r_4}}) \\
& \text{\# constraints forbidding robot co-location} \\
& \wedge \Box \bigvee_{i=1}^{4} (\varphi^c_{1_{r_i}} \wedge \bigvee_{j \neq i} \varphi^c_{2_{r_j}})
\end{aligned}
$$

Formula φ_{motion} is added as a sub formula of φ^t_s, i.e. the system safety condition.

2) Sensor Assumptions: In addition, sensor assumptions are required to define the effects of the robot activating its various motion controllers:

$$\varphi_{completion} = \bigwedge_{i,r,r'} \Box(\varphi^c_{i_r} \wedge \varphi_{i_{r'}} \Rightarrow (\bigcirc \varphi^c_{i_r} \vee \bigcirc \varphi^c_{i_{r'}}))$$

This says that if robot i is in region r (i.e. $\varphi^c_{i_r}$ is true) and is activating the controller to move to region r' (i.e. $\varphi_{i_{r'}}$ is true), then in the next time step it will either still be in r ($\varphi^c_{i_r}$ will still true), or will have arrived at r' ($\varphi^c_{i_{r'}}$ will be true). Formula $\varphi_{completion}$ is added as a sub formula of φ^t_e, i.e. the environment safety assumption.

3) Fairness conditions: In addition to the above safety conditions, further constraints on the environment are required to ensure that every motion eventually completes, i.e. that the robots' environment is in some sense "fair". In this work, we make the environment assumption that every controller activation eventually results in completion, i.e. the fairness condition (added as a sub formula of φ^g_e):

$$\varphi_{fair} = \bigwedge_{i,r} \Box\Diamond(\pi_{i_r} \Rightarrow \bigcirc \pi^c_{i_r})$$

This corresponds to the assumption that every atomic motion controller will eventually complete execution. In other words, if robot i is driving to region r, it will eventually cross the threshold into r.

Given a task specification $\varphi = (\varphi_e \Rightarrow \varphi_s)$, the LTL specification used to synthesize a controller (after adding multi-robot motion constraints, sensor assumptions and fairness conditions) is now :

$$\varphi_{new} = \; \varphi_e \wedge \varphi_{completion} \wedge \varphi_{fair} \Rightarrow \varphi_s \wedge \varphi_{motion}$$

Note that the approach presented in this work can also be used with single-robot feedback controllers such as those described in [3], [7], but the safety restrictions on the relative locations of the robots would have to be explicitly encoded in the user-defined specification.

D. Synthesis

An LTL formula φ is *realizable* if there exists a finite state strategy that, for every finite sequence of truth assignments to the sensor propositions, provides an assignment to the robot propositions such that every infinite sequence of truth assignments generated in this manner satisfies φ. The synthesis problem is to find a deterministic finite state automaton (if one exists) that encodes this strategy, i.e. whose executions correspond to sequences of truth assignments that satisfy φ. Although synthesizing a deterministic automaton that realizes an arbitrary LTL formula is doubly exponential in the size of the formula, when restricted to formulas of the form $\varphi_e \Rightarrow \varphi_s$ described above, the algorithm introduced in [14] permits synthesis in time polynomial in the size of the state space. This synthesis algorithm was extended in [15] to efficiently accommodate a wider class of specifications, and this paper uses the synthesis algorithm from that work.

When a specification is realizable, synthesis yields an automaton that implements the specification in a discrete abstraction of the problem. Fig. 2 depicts an excerpt of the automaton synthesized for Example 1 using the GR(1) synthesis algorithm in [15], as implemented in the SLUGS synthesis tool[1]; the full automaton has 204 states. Each state of the automaton is labeled with the truth assignment to location and action propositions in that state, and each transition is labeled with the truth assignment to sensor propositions required for that transition to be enabled. Incoming transitions therefore also determine the truth value of the sensor propositions for each state. In state q_0, robot 1

[1]available at https://github.com/ltlmop/slugs

is in r_3 (as indicated by incoming edge label c_13), activating the controller for moving to r_1, and is carrying metal. Robot 2 is in r_2, moving towards r_4. States q_1, q_2 and q_3 are the result of various combinations of motion-completion events, e.g. in q_3, both robot motions have completed simultaneously (we do not exclude this possibility here, although the atomic controllers in [11] preclude it).

E. Continuous Execution

If an automaton is obtained, a controller that implements the corresponding continuous behavior is constructed by viewing the automaton as a hybrid controller, with a transition between two states achieved by the activation of one or more low-level continuous controllers corresponding to each proposition. The atomic controllers used must satisfy the bisimulation property [1], which ensures that every change in the discrete robot model can be implemented in the continuous domain. Since the task specification included a discrete transition graph representing the availability of these atomic controllers, every change in the discrete multi-robot model can by design be implemented in the continuous domain (i.e., the atomic motion controllers constructed as in [11] are guaranteed to drive each robot from one region to another regardless of the initial state within the region).

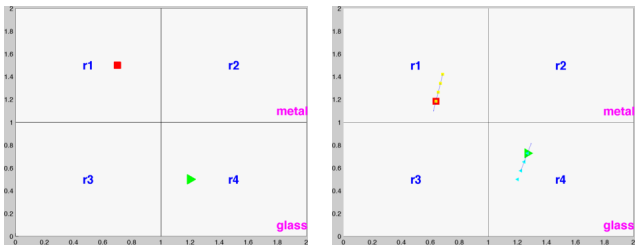
Given an automaton synthesized using the above approach, all relevant controllers are invoked simultaneously to bring about an instantaneous change in state. A given discrete transition (q, q') is executed by simultaneously invoking the controllers corresponding to every action or location proposition that changes value from q to q'. Transitions in the automaton are instantaneous, as they correspond to activation or completion of controllers. Recall that in this work, the non-motion controllers are assumed to have instantaneous execution, and so activation and completion are captured by a single event. The motion controllers may however take several discrete transitions to complete execution; this allows other events to occur while a robot is completing a motion.

IV. Simulations

In this section, we show the results of simulating the controller constructed for Example 1, and discuss differences of the proposed approach from that in [11]. Fig. 3 shows a MATLAB simulation of the automaton synthesized for Example 1; a video accompanies this paper. The naive controllers used for simulating motion for each robot set the robot's velocity towards the centroid of the next region – in future work, these will be replaced in physical experiments with atomic controllers such as those constructed in [2].

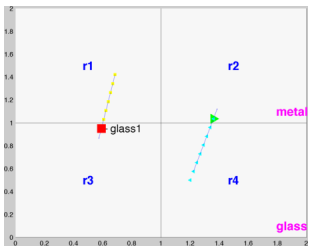
In the simulation, the robots 1 and 2 are represented by a red square and green triangle respectively. Larger markers on the trajectories indicate points where the robots change state, either by moving to a new region, or as the result of a sensor event. Fig 3(a) shows the robot initial condition, with robot 1 in r_1 and robot 2 in r_4. In Fig 3(b), the robots have both started moving towards their goals r_4 and r_1 respectively: robot 1 moves through r_3 and robot 2 moves through r_2. In Fig 3(c), robot senses glass while it is in r_3, as marked. In Fig 3(d), robot 1 enters r_4 after sensing glass, and then senses metal. In Fig 3(e), robot 1 arrives in r_2 after sensing metal (note that robot 2 has moved to r_4 to avoid being in the same room as robot 1). In Fig 3(f) the robots have visited their respective goals (robot 1 is in r_4, robot 2 is in r_1).

Observe that the robot trajectories never collide, and the robots are never in the same region at the same time, since this was the constraint added based on the available low-level atomic controllers, and enforced in the synthesized automaton. Additionally, the robots move simultaneously when they can do so without arriving in the same room (e.g. in Fig 3(b)), thus allowing progress when possible while still avoiding collisions.

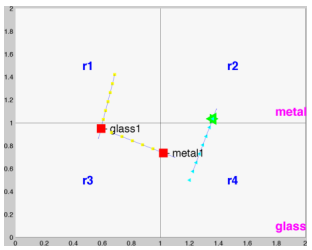


(a) Initially robot 1 is in r_1, robot 2 is in r_4

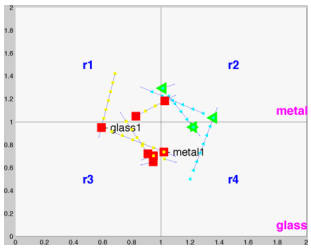
(b) Robots 1 and 2 start moving towards r_4 and r_1 via r_3 and r_2.



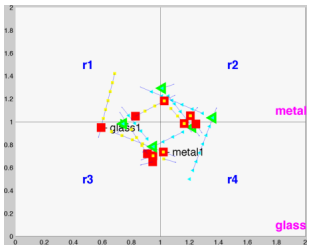
(c) Robot 1 senses glass while in r_3



(d) Robot 1 senses metal as it enters r_4 after sensing glass



(e) Robot 1 arrives in r_2 after sensing metal. Robot 2 moves aside to r_4.



(f) Robots continue to their destinations (robot 1 in r_4, robot 2 in r_1)

Fig. 3: Simulation of Example 1

A. Comparison with previous approach

By allowing more than one robot to make progress at a time, the proposed approach succeeds in some cases where requiring that at most one robot move at a time is too conservative. The following example demonstrates one such case, where restricting simultaneous motion makes the specification unsynthesizable, when in fact a controller exists to achieve the desired behavior.

Example 2 *Consider the workspace depicted in Fig 1(b). Robot 1 starts in r_1. When it sees glass, it should immediately*

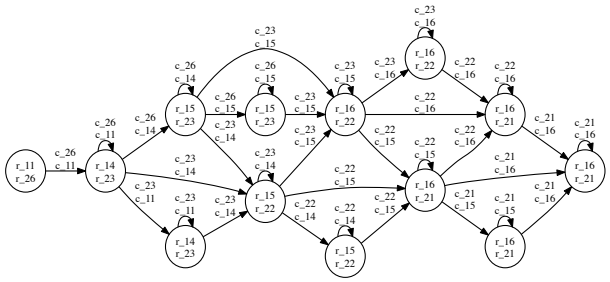


Fig. 4: Synthesized automaton for Example 2

move to r_2, and return to r_1 immediately when it stops sensing glass. Robot 2 starts in r_6, and its goal is to visit r_4.

It should be possible to synthesize a controller for the specification in 2, because the two robots operate in disjoint parts of the workspace – robot 1 only needs to move between regions r_1 and r_2, and robot 2 only needs to move between r_4, r_5 and r_6. However, using the approach in [11], each robot's motion is abstracted to a single discrete event, and only one robot is allowed to move at a time because of the nature of the constructed atomic controllers. Consider what happens if the glass sensor is toggled on and off at each successive time step. In order to satisfy the safety specification, robot 1 has to move between rooms r_1 and r_2 infinitely. However, robot 2 has to stay in place in r_6 while robot 1 is moving, and can therefore never visit r_4.

On the other hand, if we model motion activation and completion as separate events, the restriction changes from requiring only one robot to move at a time, to an assumption stating that no two robots will complete execution of a motion controller at the exact same time. Thus, with the approach presented in this paper, robot 2 can initiate motion towards r_5 and eventually r_4, even as robot 1 continues to move between r_1 and r_2. Moreover, while the two robots' motion controllers may never complete at the exact same time, they will both eventually complete (according to the fairness assumption made while synthesizing the controller). Fig 4 depicts the automaton synthesized for this specification.

B. Added complexity

With the above discrete abstraction of the multi-robot mission planning problem, one environment proposition must be added for each robot motion (corresponding to the sensor for motion completion). The strategy synthesis scales exponentially in the number of sensor propositions[2], i.e. by a factor of $2^{|\mathcal{Y}|}$. Thus, the added richness of the discrete abstraction, which allows synthesis for a wider range of task specifications, comes at the expense of an increased state space, and therefore a higher computational cost of synthesis; the synthesis algorithm itself is still polynomial in the size of the state space and therefore tractable in practice.

V. Conclusions

This paper presented a method for constructing provably correct controllers for teams of robots performing complex high-level tasks. The approach used a discrete abstraction to synthesize an automaton satisfying a task specification, and a set of atomic controllers for continuously implementing every discrete transition in the synthesized automaton. It relaxed assumptions on the simultaneity and interleaving of motion of the individual robots, while preserving safety of the resulting controllers. Allowing multiple robots to progress simultaneously overcomes inefficiencies resulting from robots awaiting their turn to move, and enables provably correct controller synthesis in cases that were previously unsynthesizable. The method was demonstrated in simulation through the example of a team of robots engaged in a recycling task, but is general, and lends itself to any multi-agent domain for which the relevant low-level atomic controllers can be created. Moreover, once the atomic controllers have been created for a specific domain, they can be reused for a wide variety of tasks by easily modifying the specification. Experimental evaluation with atomic controllers constructed using a variety of approaches is a topic of future work.

[2]This can be reduced in practice by encoding mutually exclusive actions and sensors using bit vectors.

References

[1] Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere, and George J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[2] Nora Ayanian and Vijay Kumar. Decentralized feedback controllers for multi-agent teams in environments with obstacles. *IEEE Transactions on Robotics*, 26(5):878 – 887, October 2010.

[3] Calin Belta, Volkan Isler, and George J. Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–874, 2005.

[4] Amit Bhatia, Lydia E. Kavraki, and Moshe. Y. Vardi. Sampling-based motion planning with temporal goals. In *ICRA*, pages 2689–2696, 2010.

[5] Yushan Chen, Xu Chu Ding, Alin Stefanescu, and Calin Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, 2012.

[6] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[7] David C. Conner, Alfred Rizzi, and Howie Choset. Composition of local potential functions for global robot control and navigation. In *IROS*, volume 4. IEEE, October 2003.

[8] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *CDC*, 2009.

[9] Marius Kloetzer and Calin Belta. Temporal logic planning and control of robotic swarms by hierarchical abstractions. *IEEE Transactions on Robotics*, 23(2):320–330, 2007.

[10] Marius Kloetzer. and Calin Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics*, 26(1):48–61, 2010.

[11] Hadas Kress-Gazit, Nora Ayanian, George J. Pappas, and Vijay Kumar. Recycling controllers. In *CASE*, pages 772–777, 2008.

[12] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[13] Savvas G. Loizou and Kostas J. Kyriakopoulos. Automatic synthesis of multiagent motion tasks based on ltl specifications. In *CDC*, pages 153–158, 2004.

[14] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, pages 364–380. Springer, 2006.

[15] Vasumathi Raman, Nir Piterman, and Hadas Kress-Gazit. Provably correct continuous control for high-level robot behaviors with actions of arbitrary execution durations. In *ICRA*, pages 4075–4081, 2013.

[16] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *HSCC*, pages 101–110, 2010.