Autonomous Robots (2020) 44:585–600
https://doi.org/10.1007/s10514-019-09861-4

Automated synthesis of decentralized controllers for robot swarms from high-level temporal logic specifications

Salar Moarref[1] · Hadas Kress-Gazit[1]

Received: 7 May 2018 / Accepted: 7 May 2019 / Published online: 21 May 2019

Abstract
The majority of work in the field of swarm robotics focuses on the bottom-up design of local rules for individual robots that create emergent swarm behaviors. In this paper, we take a top-down approach and consider the following problem: how can we specify a desired collective behavior and automatically synthesize decentralized controllers that can be distributed over robots to achieve the collective objective in a provably correct way? We propose a formal specification language for the high-level description of swarm behaviors on both the *swarm* and *individual* levels. We present algorithms for automated synthesis of *decentralized* controllers and *synchronization skeletons* that describe how groups of robots must coordinate to satisfy the specification. We demonstrate our proposed approach through an example in simulation.

Keywords Formal methods · Automated synthesis · Robotic swarm · Temporal logic

1 Introduction

Swarm robotics is the study of robotic systems consisting of large numbers of robots whose local interactions with each other and with their environment lead to a collectively intelligent behavior. Swarm robotic systems have many potential applications such as exploration, surveillance, search and rescue, intrusion detection, inspection, construction and cleaning. Thus, it is not surprising that swarm robotics have been a very active research area [see (Brambilla et al. 2013] for an excellent review of the literature). However, as observed in Brambilla et al. (2013), the intuition of the human designer is still the main ingredient in the development of swarm robotic systems, and a trial and error process based on iterative design and testing is an essential part of many existing design methods.

Advancements in technology are enabling mass production of cheap and capable robots, and there will be an indispensable need for scientific and engineering methods for formal mission specification and automated design techniques that result in swarm robotic systems with provably-correct collective behavior. To this end, in this paper we consider the problem of automated synthesis from high-level temporal logic specifications of decentralized controllers for safe navigation of robotic swarms. The proposed framework facilitates the design and deployment process for swarm robotic systems, while providing formal guarantees on fulfillment of the collective objective.

As a motivating example, consider a room that is partitioned into regions $\{A, B, C, D, E\}$ as shown in Fig. 1a. Assume that an operator wants (1) the whole swarm to repeatedly (*infinitely often*) gather in region A, i.e., *all* robots must be present together in region A, (2) all robots to visit region E infinitely often, but they can do so at different times, i.e., it is acceptable if a subset of the swarm visits E at time t_1 and the rest of the robots visit E at time $t_2 \neq t_1$, (3) a *part* of the swarm (i.e., at least one robot) to infinitely often visit region B, (4) the swarm to never occupy regions B and E simultaneously, and (5) no robot to enter region D.

This is one of the several papers published in *Autonomous Robots* comprising the Special Issue on Multi-Robot and Multi-Agent Systems.

This research was supported by DARPA N66001-17-2-4058.

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10514-019-09861-4) contains supplementary material, which is available to authorized users.

✉ Salar Moarref
sm945@cornell.edu

Hadas Kress-Gazit
hadaskg@cornell.edu

1 Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, USA

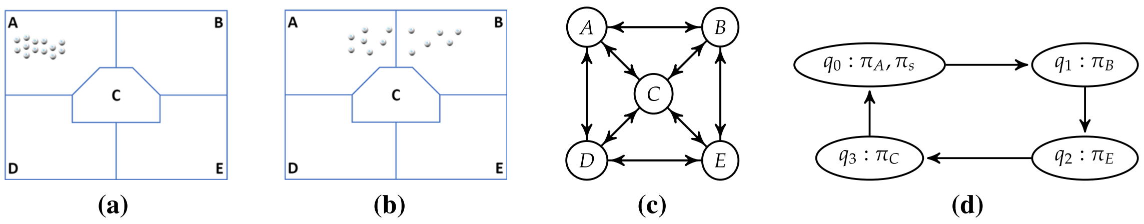

Fig. 1 **a** Workspace, **b** group of robots moving from A to B, **c** region graph and **d** labeled transition system

We distinguish between multi robot systems where the number of robots is known and typically each robot is treated as a unique agent, and a robotic swarm where robots are interchangeable and the number of robots might be large and even a priori unknown. We propose a formal specification language, based on Linear Temporal Logic (LTL), for specifying different navigation semantics over an abstraction of the system that is *independent* of the number of robots in the swarm. Given the user-specified global specification, we provide algorithms to automatically synthesize discrete and *decentralized* controllers that can be assigned to *groups* of robots such that the resulting system satisfies the given specification. To this end, we synthesize a centralized controller for the whole swarm and then partition the controller in order to obtain local programs that can be deployed on individual robots.

Synthesized local programs can be executed *asynchronously*, however, to enforce all objectives, it may be necessary for the robots to synchronize. For example, if it is required that the whole swarm must be in a region simultaneously, then a coordination mechanism is needed that signals the robots whether the whole swarm is currently in a particular region or not. By augmenting the decentralized controllers with information about *coordination* or *synchronization* such as when robots must synchronize with other robots and with which robots they must do so, we ensure that asynchronous and decentralized execution of the local programs satisfy the specification.

Each synthesized local program will be executed by a group of robots of variable size. We assume that each group moves (almost) together between the regions, e.g., if a group of robots are moving from region A to B, and some members of the group reach B, they wait for other group members to enter B before executing their next step. Careful attention must be paid when dealing with safety constraints because of the physical properties of moving in *groups*, e.g., if the specification forbids robots to be in both regions A and B at the same time, the discrete controller cannot have a transition between a state where the robots are in A to a state where the robots are in B since a group of robots moving from region A to B will never leave A and enter B instantaneously. Therefore, in reality, we will witness "intermediate" states, where part of the group occupies region A while the rest of it is in region B. Figure 1b shows such an intermediate state. The synthesized controllers and coordination mechanism must be able to accommodate these physical artifacts.

Related work With many potential application domains of swarm robotic systems, their design and analysis has been subject to extensive research and investigation (Brambilla et al. 2013). The most common way to develop a swarm robotic system is behavior-based design (Brambilla et al. 2013) where the individual behavior of each robot is implemented, studied and improved iteratively until the desired collective behavior is obtained (e.g., Nouyan et al. 2008; Soysal and Sahin 2005; Labella et al. 2006; Bachrach et al. 2010; Balch and Hybinette 2000). Existing automatic design methods for robotic swarms can be classified into two categories: evolutionary robotics (ER) and multi-robot reinforcement learning (RL) (Panait and Luke 2005). In the RL framework, an agent learns a behavior through trial and error interactions with an environment and through feedback from its actions. ER applies evolutionary computation techniques (Goldberg 1989) to single and multi-robot systems and is inspired by the Darwinian principle of natural selection and evolution. However, these approaches do not provide formal guarantees on the emergence of the desired collective behavior.

The synthesis problem was first recognized by Church (1962). The problem of synthesizing reactive systems from an LTL specification was considered by Pnueli and Rosner (1989), and shown to be doubly exponential in the size of the LTL formula (Rosner 1992). Bloem et al. (2012) present polynomial time algorithms for the realizability and synthesis problems for a fragment of LTL known as Generalized Reactivity (1) (GR(1)). The specification language used in this paper is based on GR(1).

The use of formal verification techniques for analyzing the emergent behaviors of robotic swarms is studied in Dixon et al. (2012), Gjondrekaj et al. (2012). In this paper, we consider *synthesizing correct-by-construction* decentralized controllers for swarms of robots. Hierarchical control and planning for systems with large number of identical components from high-level temporal logic specifications is also considered in Nilsson and Ozay (2016), Kloetzer and Belta (2006). However, these works synthesize *centralized* controllers. The problem of intermediate states due to arbitrary execution times of the actions for single and multi-robot systems are addressed in Raman et al. (2015), Raman and Kress-Gazit (2014), and it is shown how a centralized controller can be synthesized that is safe w.r.t. possible intermediate states. In this paper, we show how decentralized controllers for swarms can be synthesized that are safe w.r.t. possible intermediate states.

A framework similar to ours is proposed in Kloetzer et al. (2011) for synthesis of controllers for multi-robot systems, however, we provide a specification language and synthesis algorithms that are designed to handle *swarms* of robots. Furthermore, unlike Kloetzer et al. (2011), we do not require *strong* synchronization, i.e., requiring a group of robots to perfectly synchronize and move from one region to the next one exactly at the same time since for physical robots with possibly different speeds, ensuring strong synchronization is not practical. Our framework creates controllers for specifications that include the next operator only if the swarm satisfies the task under asynchronous execution.

Synthesis from high-level LTL specifications for multi-robot systems is considered in many recent works (see e.g., Guo and Dimarogonas 2015; Ulusoy et al. 2013; Karaman and Frazzoli 2008; Loizou and Kyriakopoulos 2005; Filippidis et al. 2012; Guo et al. 2014). In this paper we consider a fragment of LTL, with the advantage of applying *symbolic* synthesis algorithms that generally scale better in practice compared to automata-theoretic approaches (Bloem et al. 2012; Ehlers 2010). Furthermore, we develop abstractions and algorithms to synthesize decentralized controllers for swarms of robots where the number of robots are a priori unknown.

Synthesis of synchronization skeletons for concurrent programs is a classical problem in concurrency theory. However, to the best of our knowledge, the majority of the works in synthesis of synchronization for concurrent programs are focused on shared-memory settings or assume a centralized synchronization process (e.g., Emerson and Clarke 1982; Vechev et al. 2010; Manna and Wolper 1984). This setting is not suitable in our case since the assumption of having access to a centralized agent or global information is often violated in realistic scenarios. Here, we consider a decentralized architecture where groups of robots can coordinate as needed.

Contributions This paper is based on Moarref and Kress-Gazit (2017) and contains detailed explanations of the methods and results presented there. In addition, we describe situations where a specification may not be realizable due to unrealistic requirements on the robot numbers and provide a new algorithm for quantitative analysis of the centralized controller that decides if it is *feasible*, i.e., the centralized controller can be executed by a finite number of robots, and if so, computes the minimum number of robots that are required to implement the symbolic plan in a decentralized fashion (Sect. 4.2).

Organization The organization of the paper is as follows. In Sect. 2 we introduce some notation, background and definitions that are used in the rest of the paper. In Sect. 3 we show how desired swarm behavior can be specified in our proposed framework. In Sect. 4 we explain our solution. In Sect. 5 we demonstrate our algorithms over a swarm robotic example. In Sect. 6 we evaluate the performance of our framework over a set of examples. Finally, in Sect. 7 we conclude and discuss possible future directions.

2 Preliminaries

Temporal logic We use linear temporal logic (LTL) to specify system objectives. LTL is a formal specification language with two types of operators: logical connectives (e.g., $\neg$ (negation) and $\wedge$ (conjunction)) and temporal operators (e.g., $\bigcirc$ (next), $\mathcal{U}$ (until), $\Diamond$ (eventually), and $\Box$ (always)). The formulas of LTL are defined over a set of atomic propositions (Boolean variables) $\mathcal{V}$. The syntax is given by the grammar:

$$\Phi := v \mid \Phi \vee \Phi \mid \neg\Phi \mid \bigcirc \Phi \mid \Phi \,\mathcal{U}\, \Phi \text{ for } v \in \mathcal{V}$$

We define $\texttt{True} = v \vee \neg v$, $\texttt{False} = v \wedge \neg v$, $\Diamond\Phi = \texttt{True}\,\mathcal{U}\,\Phi$, and $\Box\Phi = \neg\Diamond\neg\Phi$. A formula with no temporal operator is a *predicate*. Given a predicate ϕ over variables $\mathcal{V}$, we say $s \in 2^{\mathcal{V}}$ satisfies ϕ, denoted by $s \models \phi$, if the formula obtained from ϕ by replacing all variables in s by True and all other variables by False is valid. We call the set of all possible assignments to variables $\mathcal{V}$ *symbols* and denote them by $\Sigma_{\mathcal{V}}$, i.e., $\Sigma_{\mathcal{V}} = 2^{\mathcal{V}}$. An LTL formula over variables $\mathcal{V}$ is interpreted over infinite words $w \in (\Sigma_{\mathcal{V}})^{\omega}$. The language of an LTL formula Φ, denoted by $\mathcal{L}(\Phi)$, is the set of infinite words that satisfy Φ, i.e., $\mathcal{L}(\Phi) = \{w \in (\Sigma_{\mathcal{V}})^{\omega} \mid w \models \Phi\}$. We refer the reader to Clarke et al. (2009) for a more formal introduction to LTL.

Labeled transition system (LTS) An LTS is a tuple $\mathcal{T} = \langle Q, q_0, \mathcal{V}, \delta, \mathcal{L}\rangle$ where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\mathcal{V}$ is a set of propositions, $\delta \subseteq Q \times Q$ is a transition relation, and $\mathcal{L} : Q \rightarrow 2^{\mathcal{V}}$ is a labeling function which maps each state to a set of propositions that hold in that state. A *run* of an LTS is an infinite sequence of states $q_0q_1q_2...$ where $q_i \in Q$ and $(q_i, q_{i+1}) \in \delta$ for all $i \geq 0$. The language of an LTS $\mathcal{T}$ is defined as the set $\mathcal{L}(\mathcal{T}) = \{\mathcal{L}(q_0)\mathcal{L}(q_1)\mathcal{L}(q_2)\cdots \in (2^{\mathcal{V}})^{\omega} \mid q_0q_1q_2\cdots \text{ is a run of } \mathcal{T}\}$, i.e., the set of (infinite) words generated by the runs of $\mathcal{T}$. An LTS $\mathcal{T}$ *realizes* an LTL specification Φ iff all infinite words in its language satisfy Φ, i.e., $\forall w \in \mathcal{L}(\mathcal{T}).\ w \models \Phi$. Given an LTL specification, the synthesis problem is to find an LTS that realizes it. LTSs represent the symbolic plans or controllers executed by the robots. Figure 1d shows an example of an LTS where $Q = \{q_0, q_1, q_2, q_3\}$ is the set of states, q_0 is the initial state,

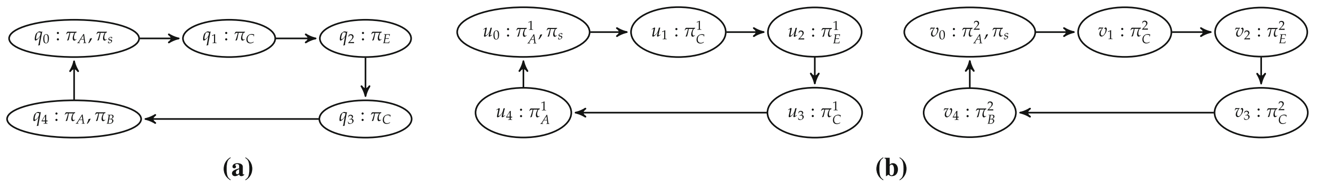

Fig. 2 **a** A centralized LTS $\mathcal{T}$. **b** $\mathcal{T}$ is partitioned into $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (right)

and $\mathcal{V} = \{\pi_A, \pi_B, \pi_C, \pi_E, \pi_s\}$ is the set of propositions. The directed edges in Fig. 1d show possible transitions between states, i.e., $\delta = \{(q_0, q_1), (q_1, q_2), (q_2, q_3), (q_3, q_0)\}$. Each state is labeled with the set of propositions that hold in that state, e.g., $\mathcal{L}(q_1) = \{\pi_B\}$.

Generalized reactivity (1) (GR(1)) GR(1) is a fragment of LTL for which *symbolic* realizability and synthesis algorithms exist (Bloem et al. 2012). GR(1) specifications are of the form $\Phi = \Phi_e \rightarrow \Phi_s$, where Φ_α for $\alpha \in \{e, s\}$ has the structure

$$\Phi_\alpha = \theta^\alpha_{\text{init}} \wedge \bigwedge_i \Box\psi^\alpha_i \wedge \bigwedge_i \Box\Diamond\phi^\alpha_i.$$

Intuitively, Φ_e indicates the assumptions on the environment and Φ_s characterizes the requirements of the system. Roughly speaking, $\theta^\alpha_{\text{init}}$ is a predicate specifying the initial conditions, ψ^α_i are predicates or LTL formulas that may only include the next temporal operator, characterizing safe, allowable moves and invariants, and ϕ^α_i are predicates that must hold repeatedly (i.e., infinitely often) during system execution, representing *liveness* assumptions and guarantees. We refer the reader to Bloem et al. (2012) for formal definitions. In Bloem et al. (2012), a symbolic synthesis algorithm for GR(1) specifications is given that performs $O(mn2^{2|\mathcal{V}|})$ symbolic steps (i.e., next step computations) in the worst case, where m is the number of liveness assumptions, n is the number of liveness guarantees, and $\mathcal{V}$ is the set of atomic propositions, respectively.

In this paper, we only consider navigation tasks and we assume that the environment is static and known, i.e., $\Phi_e =$ True. In general, GR(1) allows specification and synthesis of *reactive* controllers that respond to dynamic changes in the environment. Extension to full GR(1) is the subject of our current research.

Synchronization skeleton Let $\mathbf{P} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ be a set of LTSs. A synchronization skeleton $\mathcal{S}_i$ for LTS $\mathcal{T}_i$ with respect to $\mathbf{P}$ is a function $\mathcal{S}_i : Q_i \rightarrow 2^{\mathbf{P}}$ that maps each state $q \in Q_i$ of $\mathcal{T}_i$ to a set of LTSs $\mathcal{S}_i(q) \subseteq \mathbf{P}$ indicating that LTS_i must synchronize with LTSs $\mathcal{T}_j \in \mathcal{S}_i(q)$ before executing the next transition.

Mixed-synchronous–asynchronous composition of LTSs Given a set $\mathbf{P} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ of LTSs and their corresponding synchronization skeletons $\mathcal{S}_1, \ldots, \mathcal{S}_k$, their mixed-synchronous–asynchronous composition, denoted by $\mathcal{T}^\odot$, is an LTS itself. Roughly speaking, at each state of the composition, an LTS $\mathcal{T}_i$ is selected for execution while the other LTSs stay at the same state. If $\mathcal{T}_i$, at its current state, does not need to synchronize with any other LTSs or that the necessary synchronizations have already happened, $\mathcal{T}_i$ moves to its next state according to its transition relation. Otherwise, $\mathcal{T}_i$ remains in the same state waiting for necessary synchronizations. More specifically, $\mathcal{T}^\odot$ includes a set $\mathcal{A} \subset \mathcal{V}$, $\mathcal{A} = \{\mathbf{a}_{ij} \mid 1 \leq i, j \leq k, i \neq j\}$ of propositions where each $\mathbf{a}_{ij}$ represents whether a synchronization has happened between LTSs $\mathcal{T}_i$ and $\mathcal{T}_j$ or not (similar to acknowledge messages in networking). If two LTSs $\mathcal{T}_i$ and $\mathcal{T}_j$ synchronize, a_{ij} and a_{ji} are set to `True`. If an LTS $\mathcal{T}_i$ satisfies all synchronization requirements and is scheduled to execute, all its corresponding a_{ij}'s are reset by setting them to `False` in the next step.

For example, consider two LTSs $\mathcal{T}_1$ and $\mathcal{T}_2$ shown in Fig. 2b. Assume that $\mathcal{T}_1$ and $\mathcal{T}_2$ synchronize with each other at states u_0 and v_0, respectively. The sequence $\sigma = (u_0, v_0)(u_1, v_0)\cdots(u_4, v_0)(u_0, v_0)(u_0, v_1)(u_0, v_2)\cdots$ is a possible interleaving of states of $\mathcal{T}_1$ and $\mathcal{T}_2$ in their mixed-synchronous–asynchronous composition. Note that in σ, $\mathcal{T}_1$ and $\mathcal{T}_2$ synchronize initially, and propositions a_{12} and a_{21} are set to `True` (i.e., initially $\mathcal{T}_1$ and $\mathcal{T}_2$ are in states u_0 and v_0, respectively, with $a_{12} = a_{21} =$ `True`). $\mathcal{T}_1$ is scheduled to execute until returning to u_0 while $\mathcal{T}_2$ remains in v_0. Once $\mathcal{T}_1$ leaves the state u_0, a_{12} is reset (i.e., $\mathcal{T}_1$ moves to u_1 and a_{12} is set to `False`). Upon visiting u_0 the second time, $\mathcal{T}_1$ needs to synchronize with $\mathcal{T}_2$. Since a_{12} is `False` while a_{21} is `True`, $\mathcal{T}_1$ must wait for $\mathcal{T}_2$ to execute and come back to v_0. Then $\mathcal{T}_1$ and $\mathcal{T}_2$ can synchronize and continue their executions. Note that if two LTSs do not synchronize with each other at any state, then any interleaving of their executions is a possible run in the composition (i.e., *fully-asynchronous* composition). On the other hand, if two LTSs synchronize with each other at every state (i.e., *fully-synchronous* composition), each LTS after executing a step must wait for the other one to catch up and synchronize before being able to proceed, e.g., $(u_0, v_0)(u_1, v_0)(u_1, v_1)(u_1, v_2)(u_2, v_2)\cdots$ is a possible run.

3 Specification of swarm behaviors

We define a specification language that captures desired swarm behaviors. In our framework, the user provides a *region graph* that represents the connectivity of the workspace. They also provide a temporal logic specification describing the objectives of the system.

Region graph Let $\mathbf{R} = \{\mathbf{r}_1, \ldots, \mathbf{r}_k\}$ be a set of regions partitioning the workspace. The user provides a region graph $\mathbf{G_R} = (\mathbf{R}, \mathbf{E})$ where the vertices $\mathbf{R}$ are the regions and $\mathbf{E} \subseteq \mathbf{R} \times \mathbf{R}$ is the set of possible transitions between regions, i.e., $(\mathbf{r}_i, \mathbf{r}_j) \in \mathbf{E}$ indicates that robots can move from region $\mathbf{r}_i$ to $\mathbf{r}_j$. Figure 1c shows the region graph corresponding to the workspace in Fig. 1a. We assume that $\forall \mathbf{r} \in \mathbf{R}.\ (\mathbf{r}, \mathbf{r}) \in \mathbf{E}$, i.e., robots can stay in a region. We do not show the self-loops over the region graph to keep the figures of region graphs simple.

Region propositions The user specifies the navigation behaviors over an abstraction obtained from the region graph $\mathbf{G_R}$ where each region is represented by a unique proposition. Formally, let $\pi_\mathbf{r}$ be a proposition that is true iff a *part* of the swarm is currently in region $\mathbf{r}$. To simplify the notation, we introduce a predicate $\pi_\mathbf{r}^{\text{full}} = \pi_\mathbf{r} \wedge \bigwedge_{\mathbf{r}' \in \mathbf{R} \setminus \{\mathbf{r}\}} \neg \pi_{\mathbf{r}'}$ that holds iff some of the robots are in $\mathbf{r}$ and there are no robots in any other region, i.e., the whole swarm is in region $\mathbf{r}$. We let $\Pi = \{\pi_{\mathbf{r}_1}, \ldots, \pi_{\mathbf{r}_k}\}$ be the set of all propositions corresponding to the regions.

Moreover, to distinguish between when we talk about groups or *individual* robots, we introduce a (parametric) proposition for each region. Let $\pi_\mathbf{r}^\mathbf{a}$ be a proposition that is true iff an individual robot is in region $\mathbf{r}$. We define $\Pi^\mathbf{a} = \{\pi_{\mathbf{r}_1}^\mathbf{a}, \ldots, \pi_{\mathbf{r}_k}^\mathbf{a}\}$ as the set of robot region propositions. Since an individual robot cannot be at two different regions at the same time, the truth values of the robot region propositions are mutually exclusive, i.e., $\forall \pi_{\mathbf{r}_i}^\mathbf{a}, \pi_{\mathbf{r}_j}^\mathbf{a} \in \Pi^\mathbf{a}, \mathbf{r}_i \neq \mathbf{r}_j.\ \pi_{\mathbf{r}_i}^\mathbf{a} \rightarrow \neg \pi_{\mathbf{r}_j}^\mathbf{a}$. Note that in the following we quantify over $\mathbf{a}$; therefore, in practice, we are not actually creating propositions for each robot.

Specification We allow the user to specify the desired behaviors as a temporal logic specification at two levels: a *macroscopic* specification Φ^M that describes how *groups* of robots should behave and is given as a GR(1) specification over region propositions Π, and a *microscopic* specification Φ^μ that describes how *individual* robots must behave and is given as

$$\Phi^\mu = \bigwedge_i \forall \mathbf{a}.\ \Box \phi_i^\mathbf{a} \wedge \bigwedge_j \forall \mathbf{a}.\ \Box\Diamond \phi_j^\mathbf{a} \qquad (1)$$

where $\phi_i^\mathbf{a}$ and $\phi_j^\mathbf{a}$ are predicates over robot region propositions $\Pi^\mathbf{a}$. For example, $\forall \mathbf{a}.\ \Box(\neg \pi_{\mathbf{r}_i}^\mathbf{a} \wedge \neg \pi_{\mathbf{r}_j}^\mathbf{a})$ requires all robots to avoid regions $\mathbf{r}_i$ and $\mathbf{r}_j$ at all times.

Example 1 The example introduced in Sect. 1 can be formally specified as follows. A macroscopic specification $\Phi^M = \Box\Diamond(\pi_A^{\text{full}}) \wedge \Box\Diamond(\pi_B) \wedge \Box(\neg \pi_D) \wedge \Box(\neg(\pi_B \wedge \pi_E))$ where the first conjunct says that the whole swarm must repeatedly meet at region A, the second indicates that part of the swarm must repeatedly visit region B, the third requires that no subswarm should enter region D, and the fourth specifies that robots must never occupy regions B and E at the same time, and a microscopic specification $\Phi^\mu = \forall \mathbf{a}.\ \Box\Diamond(\pi_E^\mathbf{a})$ indicating that all robots must repeatedly visit region E.

4 Synthesis

In this section we explain how we automatically synthesize decentralized LTSs along with their synchronization skeletons that realize the given specification.

4.1 Synthesis of a centralized solution

To synthesize decentralized controllers, we first synthesize a centralized LTS that satisfies the input specification. To this end, we create a GR(1) specification Ψ from the macroscopic specification Φ^M and microscopic specification Φ^μ. We define $\pi_s \notin \Pi$ to be a *synchronization* proposition. Intuitively, π_s holds at a state if (some) groups of robots must synchronize. The macroscopic specification Φ^M is automatically translated into a GR(1) specification Ψ^M as follows. If a safety formula $\Box\varphi$ appears in Φ^M, it is also added to Ψ^M as is. For each liveness formula $\Box\Diamond\varphi$ that appears in Φ^M, we add $\Box\Diamond(\varphi \wedge \pi_s)$ to Ψ^M. Intuitively, $\Box\Diamond(\varphi \wedge \pi_s)$ indicates that (some) groups of robots may need to synchronize to satisfy the global liveness requirement, e.g., if all the robots must infinitely often be at the same region.

Next, we translate Φ^μ into a specification Ψ^μ. Intuitively, this is done by removing the universal quantifiers and replacing the predicates $\phi^\mathbf{a}$ in sub-formulas of Φ^μ by corresponding predicates $\psi^{\phi^\mathbf{a}}$ defined over Π. We first give an example.

Example 2 Consider the workspace shown in Fig. 1a. The formula $\forall \mathbf{a}.\ \Box(\phi_1^\mathbf{a}) = \forall \mathbf{a}.\ \Box(\neg \pi_D^\mathbf{a} \wedge \neg \pi_E^\mathbf{a})$ (i.e., no robot must enter D and E) is translated into $\Box(\psi^{\phi_1^\mathbf{a}}) = \Box(\neg \pi_D \wedge \neg \pi_E)$ (i.e., no part of swarm must occupy regions D and E). Similarly, the formula $\forall \mathbf{a}.\ \Box\Diamond(\phi_2^\mathbf{a}) = \forall \mathbf{a}.\ \Box\Diamond(\pi_A^\mathbf{a} \vee \pi_B^\mathbf{a})$ (i.e., all robots are required to repeatedly visit regions A or B) is translated into $\Box\Diamond(\psi^{\phi_2^\mathbf{a}}) = \Box\Diamond(\neg \pi_C \wedge \neg \pi_D \wedge \neg \pi_E)$ (i.e., the swarm must repeatedly *not* be in C, D and E).

Note that since an individual robot cannot be at two regions at the same time, the truth values of the propositions $\Pi^\mathbf{a}$ are mutually exclusive. Formally, let $\phi^{mutex} = \bigwedge_{\mathbf{r} \in \mathbf{R}} (\pi_\mathbf{r}^\mathbf{a} \rightarrow \bigwedge_{\mathbf{r}' \in \mathbf{R} \setminus \{\mathbf{r}\}} \neg \pi_{\mathbf{r}'}^\mathbf{a})$ be a predicate indicating that if a robot $\mathbf{a}$ is in region $\mathbf{r}$, then it cannot be in any other region $\mathbf{r}' \in$

$\mathbf{R}\backslash\{\mathbf{r}\}$. Now consider a predicate $\phi^{\mathbf{a}}$ over $\Pi^{\mathbf{a}}$. First, observe that the set $\Sigma^{\phi^{\mathbf{a}}} \subseteq 2^{\Pi^{\mathbf{a}}}$ of possible truth assignments to propositions in $\Pi^{\mathbf{a}}$ that satisfies ϕ and also ϕ^{mutex} is of the form $\Sigma^{\phi^{\mathbf{a}}} = \{\{\pi_{\mathbf{r}}^{\mathbf{a}}\} \in 2^{\Pi^{\mathbf{a}}} \mid \{\pi_{\mathbf{r}}^{\mathbf{a}}\} \models \phi \wedge \phi^{mutex}\}$, i.e., $\Sigma^{\phi^{\mathbf{a}}}$ is either empty or consists of singleton sets that assign `True` to exactly one robot region proposition (due to the mutual exclusion condition), e.g., a predicate $\phi^{\mathbf{a}} = \neg\pi_A^{\mathbf{a}} \wedge \neg\pi_B^{\mathbf{a}}$ is satisfied if the robot **a** occupies any region other than A or B, i.e., if exactly one proposition $\pi_{\mathbf{r}}^{\mathbf{a}} \in \Pi^{\mathbf{a}}\backslash\{\pi_A^{\mathbf{a}}, \pi_B^{\mathbf{a}}\}$ holds.

Let $\mathbf{R}^{\phi^{\mathbf{a}}} = \{\mathbf{r} \in \mathbf{R} \mid \{\pi_{\mathbf{r}}^{\mathbf{a}}\} \in \Sigma^{\phi^{\mathbf{a}}}\}$ be the set of regions that an individual robot can occupy to satisfy $\phi^{\mathbf{a}} \wedge \phi^{mutex}$. The microscopic specification ϕ^{μ} as defined in (1) is automatically translated into a GR(1) specification

$$\Psi^{\mu} = \bigwedge_i \Box\psi^{\phi_i^{\mathbf{a}}} \wedge \bigwedge_j \Box\Diamond\psi^{\phi_j^{\mathbf{a}}}$$

where $\psi^{\phi_l^{\mathbf{a}}} = \bigwedge_{\mathbf{r}\in\mathbf{R}\backslash\mathbf{R}^{\phi_l^{\mathbf{a}}}} \neg\pi_r$ is a predicate over Π corresponding to $\phi_l^{\mathbf{a}}$. Assuming that no robot can be at two regions at the same time, the safety formulas in ϕ^{μ} are "semantically equivalent" to their corresponding formulas in ψ^{μ}. Note that a safety formula $\forall\mathbf{a}.\ \Box(\phi^{\mathbf{a}})$ in ϕ^{μ} holds iff all robots are in any region $\mathbf{r} \in \mathbf{R}^{\phi^{\mathbf{a}}}$ at all times, or equivalently, iff no robot (no part of swarm) is in any region $\mathbf{r} \in \mathbf{R}\backslash\mathbf{R}^{\phi^{\mathbf{a}}}$, i.e., $\Box(\psi^{\phi^{\mathbf{a}}}) = \Box(\bigwedge_{\mathbf{r}\in\mathbf{R}\backslash\mathbf{R}^{\phi^{\mathbf{a}}}} \neg\pi_r)$ holds.

On the other hand, ψ^{μ} may impose a stronger requirement on the swarm as it requires the swarm to visit particular regions *at the same time* to satisfy a liveness property. For example, the formula $\forall\mathbf{a}.\ \Box\Diamond(\pi_E^{\mathbf{a}})$ is translated into $\Box\Diamond(\neg\pi_A \wedge \neg\pi_B \wedge \neg\pi_C \wedge \neg\pi_D) = \Box\Diamond(\pi_E^{full})$, requiring the whole swarm to be in E repeatedly and at the same time. Thus, groups of robots may have to "wait" for each other in the centralized solution, however, note that the robots do *not* need to synchronize and those unnecessary waitings (i.e., staying in the same region) are removed during the decentralization process. This approximation allows us to synthesize an LTS that ensures that all the robots repeatedly visit a particular region possibly at different times.

Remark 1 In Moarref and Kress-Gazit (2017), the microscopic specification Φ^{μ} is translated into a specification Ψ^{μ} by removing the universal quantifier and replacing the individual robot propositions $\pi_{\mathbf{r}}^{\mathbf{a}} \in \Pi^{\mathbf{a}}$ with their corresponding *full region predicate* $\pi_{\mathbf{r}}^{full}$, e.g., the formula $\forall\mathbf{a}.\ \Box\Diamond(\pi_{\mathbf{r}}^{\mathbf{a}})$ is translated into $\Box\Diamond(\pi_{\mathbf{r}}^{\text{full}})$. This translation works if the predicates $\phi_i^{\mathbf{a}}$ and $\phi_j^{\mathbf{a}}$ in (1) are written in *positive normal form*, i.e., no negation symbol appears in the predicates. For example, consider the formula $\forall\mathbf{a}.\Box\Diamond(\phi^{\mathbf{a}})$ where $\phi^{\mathbf{a}} = \neg\pi_A^{\mathbf{a}} \wedge \neg\pi_B^{\mathbf{a}}$, requiring all the agents to *not* be in regions A and B infinitely often. Note that $\phi^{\mathbf{a}}$ is not in positive normal form, and the formula $\Box\Diamond(\neg\pi_A^{full} \wedge \neg\pi_B^{full})$ does not imply $\forall\mathbf{a}.\Box\Diamond(\phi^{\mathbf{a}})$. The translation proposed in this paper resolves this issue and can treat general Boolean formulas defined over $\Pi^{\mathbf{a}}$.

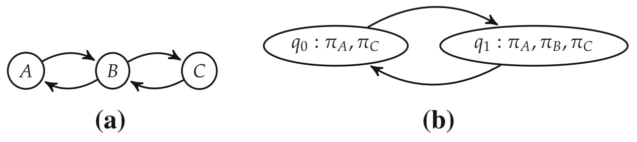

Fig. 3 **a** Region graph, and **b** an LTS

Finally, we define $\Psi = \Psi^M \wedge \Psi^{\mu}$ to be the specification created using the user-provided macroscopic and microscopic specifications.

Example 3 The specification in Example 1 is translated into $\Psi^M = \Box\Diamond(\pi_A^{\text{full}} \wedge \pi_s) \wedge \Box\Diamond(\pi_B) \wedge \Box(\neg\pi_D) \wedge \Box(\neg(\pi_B \wedge \pi_E))$, and $\Psi^{\mu} = \Box\Diamond(\neg\pi_A \wedge \neg\pi_B \wedge \neg\pi_C \wedge \neg\pi_D)$.

4.1.1 Specification of region graph

To complete the specification, we also need to formally specify the transition rules between regions that represent the feasible motion of the robots. Formally, for each $\mathbf{r} \in \mathbf{R}$, let $\mathbf{outgoing}(\mathbf{r}) = \{\mathbf{r}' \in \mathbf{R} \mid (\mathbf{r}, \mathbf{r}') \in \mathbf{E}\}$ and $\mathbf{incoming}(\mathbf{r}) = \{\mathbf{r}' \in \mathbf{R} \mid (\mathbf{r}', \mathbf{r}) \in \mathbf{E}\}$ be region $\mathbf{r}$'s corresponding set of outgoing and incoming neighbors, respectively. For each $\mathbf{r} \in \mathbf{R}$ we write two LTL formulas $\Phi_{\mathbf{r}} = \Box(\pi_{\mathbf{r}} \rightarrow \bigvee_{\mathbf{r}'\in\mathbf{outgoing}(\mathbf{r})} \bigcirc(\pi_{\mathbf{r}}'))$ and $\Phi_{\mathbf{r}'} = \Box(\bigcirc(\pi_{\mathbf{r}}) \rightarrow \bigvee_{\mathbf{r}'\in\mathbf{incoming}(\mathbf{r})} \pi_{\mathbf{r}'})$. Intuitively, $\Phi_{\mathbf{r}}$ specifies that if a part of swarm is in region $\mathbf{r}$, then at the next step the robots can occupy any of its adjacent regions or stay in region $\mathbf{r}$, while $\Phi_{\mathbf{r}'}$ encodes that if a region $\mathbf{r}$ is occupied at the next step, then $\mathbf{r}$ or at least one of its neighbors is occupied in the current step. Finally, the specification $\Phi_{\mathbf{G_R}}$ corresponding to the region graph $\mathbf{G_R}$ is the conjunction of formulas for each region, i.e., $\Phi_{\mathbf{G_R}} = \bigwedge_{\mathbf{r}\in\mathbf{R}} \Phi_{\mathbf{r}} \wedge \Phi_{\mathbf{r}'}$.

Example 4 Consider the region graph $\mathbf{G_R}$ shown in Fig. 3a. It can be formally specified as conjunction of the following formulas:

- $\Phi_{\mathbf{r}} = \Box(\pi_{\mathbf{r}} \rightarrow \bigcirc(\pi_{\mathbf{r}} \vee \pi_B))$ for $\mathbf{r} \in \{A, C\}$,
- $\Phi_{\mathbf{r}'} = \Box(\bigcirc(\pi_{\mathbf{r}}) \rightarrow (\pi_{\mathbf{r}} \vee \pi_B))$ for $\mathbf{r} \in \{A, C\}$,
- $\Phi_B = \Box(\pi_B \rightarrow \bigcirc(\pi_A \vee \pi_B \vee \pi_C))$, and
- $\Phi_{B'} = \Box(\bigcirc(\pi_B) \rightarrow (\pi_A \vee \pi_B \vee \pi_C))$,

The specification $\Phi_{\mathbf{G_R}}$ is conjoined with Ψ obtained from the macroscopic and microscopic specifications to create a new specification $\Phi^C = \Psi \wedge \Phi_{\mathbf{G_R}}$. A centralized LTS is then synthesized, using the synthesis algorithm proposed in Bloem et al. (2012), that realizes Φ^C.

Example 5 Consider the specification in Example 3. A centralized LTS $\mathcal{T}$ is synthesized that moves the swarm from region A to B, then E, then C, and then back to A, and this is repeated indefinitely as shown in Fig. 1d. Note that region D is avoided as it is required by the specification.

4.1.2 Intermediate states

In the synthesized LTS, the robots can move between the regions "instantaneously", i.e., in a single step. For example, in the LTS shown in Fig. 1d, the swarm moves from region B to E in a single step. However, one of the characteristics of a group moving from one region to the next is that instantaneous region transitions are impossible, thus the execution goes through "intermediate" states where the robots are spread out between two regions. In this work we check if the intermediate states resulting from transitions of the synthesized LTS $\mathcal{T}$ can violate any safety guarantee.

To this end, for each transition $\tau = (q_i, q_{i+1}) \in \delta$ of the synthesized LTS $\mathcal{T} = (Q, q_0, \mathcal{V}, \delta, \mathcal{L})$ a predicate ϕ_τ is obtained that encodes all possible intermediate states that can occur during execution of that transition, as follows. Let $\pi_\mathbf{r} \in \mathcal{L}(q_i)$ be a region proposition that hold in state q_i, and let $\gamma_\mathbf{r}^{q_i} = \{\mathbf{r}' \in \mathbf{R} \mid \pi_{\mathbf{r}'} \in \mathcal{L}(q_{i+1}) \text{ and } (\mathbf{r}, \mathbf{r}') \in \mathbf{E}\}$ be the set of regions r' such that $\pi_{\mathbf{r}'}$ holds in the next state q_{i+1} and there is an edge from $\mathbf{r}$ to $\mathbf{r}'$ in $\mathbf{G_R} = (\mathbf{R}, \mathbf{E})$. For each $\pi_\mathbf{r} \in \mathcal{L}(q_i)$, we define a predicate

$$\phi_\mathbf{r} = \bigwedge_{\mathbf{r}' \in \gamma_\mathbf{r}^{q_i}} (\pi_\mathbf{r} \vee \pi_{\mathbf{r}'})$$

that encodes possible states that may be visited due to transition between regions $\mathbf{r}$ and any region $\mathbf{r}' \in \gamma_\mathbf{r}^{q_i}$. Intuitively, the disjunct $\pi_\mathbf{r} \vee \pi_{\mathbf{r}'}$ in $\phi_\mathbf{r}$ indicates that while a group of robots are moving from $\mathbf{r}$ to $\mathbf{r}'$, region $\mathbf{r}$ or $\mathbf{r}'$ or both may be occupied while the transition is being executed. let $\beta = \Pi \backslash (\mathcal{L}(q_i) \cup \mathcal{L}(q_{i+1})$ be the set of "uninvolved" region propositions, i.e., regions that are not occupied in either source q_i or destination q_{i+1} states. The set of all intermediate states that may be visited during the transition $\tau = (q_i, q_{i+1}) \in \delta$ is represented symbolically by the predicate

$$\phi_\tau = \left(\bigwedge_{\pi_\mathbf{r} \in \mathcal{L}(q_i)} \phi_\mathbf{r} \right) \wedge \left(\bigwedge_{\pi_{\mathbf{r}'} \in \beta} \neg \pi_{\mathbf{r}'} \right) \wedge \neg \phi_{q_i} \wedge \neg \phi_{q_{i+1}}$$

where

$$\phi_\theta = \left(\bigwedge_{\pi_\mathbf{r} \in \mathcal{L}(\theta)} \pi_\mathbf{r} \right) \wedge \left(\bigwedge_{\pi_{\mathbf{r}'} \in \Pi \backslash \mathcal{L}(\theta)} \neg \pi_{\mathbf{r}'} \right) \quad \text{for } \theta \in \{q_i, q_{i+1}\}.$$

Intuitively, ϕ_{q_i} and $\phi_{q_{i+1}}$ are predicates representing the source q_i and destination q_{i+1} states, respectively, and are removed from the set of possible intermediate states (since they are not intermediate). The conjunct $\bigwedge_{\pi_{\mathbf{r}'} \in \beta} \neg \pi_{\mathbf{r}'}$ in ϕ_τ ensures that regions that are not occupied during the transitions, stay empty.

Let Ψ^{safe} be the safety properties of the specification Ψ obtained from Φ^μ and Φ^M. A transition τ of the LTS $\mathcal{T}$ is safe, considering possible intermediate states, if $\phi_\tau \models \Psi^{\text{safe}}$, i.e., its corresponding predicate ϕ_τ satisfies the safety requirements of the given specification. If all transitions of $\mathcal{T}$ are safe, $\mathcal{T}$ is returned as a centralized solution that satisfies the input specification and is safe w.r.t. possible intermediate states. Otherwise, the specification is strengthened by adding safety formulas that prohibits taking the unsafe transitions.

4.1.3 Ruling out an unsafe transition

The predicate ϕ_τ encodes possible intermediate states when the swarm moves from state q_i to state q_{i+1} in the LTS $\mathcal{T}$. If ϕ_τ does not satisfy the safety properties of the specification, i.e., $\phi_\tau \not\models \Psi^{\text{safe}}$, then the transition $\tau = (q_i, q_{i+1})$ is not a safe transition w.r.t. intermediate states. Therefore, the synthesized LTS τ can violate the safety requirements and an alternative LTS must be synthesized. To ensure that the new LTS cannot have the unsafe transition, we update the specification by adding a new safety rule that prohibits the unsafe transition. To this end, the safety formula $\tau = \Box(\phi_{q_i} \rightarrow \bigcirc(\neg \phi_{q_{i+1}}))$ is added to the specification that ensures that transition from regions represented by $\mathcal{L}(q_i)$ to regions represented by $\mathcal{L}(q_{i+1})$ is not allowed in the LTS synthesized in the next iteration, if such an LTS exists. Intuitively, the transition between q_i and q_{i+1} is discovered to be unsafe and ruled out from the next solution.

The iterative process of synthesizing an LTS, checking its safety w.r.t. possible intermediate states, and refining the specification if necessary, is repeated until either an LTS is synthesized that is safe w.r.t. intermediate states, or the specification becomes unrealizable, indicating that there is no solution satisfying all the constraints. Since the state space is finite, the algorithm is guaranteed to terminate.

Example 6 Consider the LTS shown in Fig. 1d. Observe that this LTS satisfies the specification given in Example 3, however, considering the intermediate states, the specification would be violated since the transition between regions B and E results in an intermediate state violating the requirement $\Box(\neg(\pi_B \wedge \pi_E))$. In other words, the transition between the state where all the robots are in region B and the state where all the robots are in region E is not safe considering the possible intermediate state, therefore our algorithm automatically adds the requirement $\Box(\phi_{q_1} \rightarrow \bigcirc(\neg \phi_{q_2})) = \Box((\pi_B \wedge \bigwedge_{\mathbf{r} \in \mathbf{R} \backslash \{B\}} \neg \pi_\mathbf{r}) \rightarrow \bigcirc(\neg(\pi_E \wedge \bigwedge_{\mathbf{r} \in \mathbf{R} \backslash \{E\}} \neg \pi_\mathbf{r})))$ to the specification, ruling out such transition and ensuring that the next computed LTS will not include it. This process of checking the synthesized LTSs and ruling out unsafe transitions is repeated until an LTS is computed that is safe w.r.t. intermediate states. For example, an alternative LTS, shown in Fig. 2a, realizing the specification moves the swarm from

Algorithm 1: Synthesis of a centralized LTS

```
Data: $\mathbf{G_R}$: a region graph , $\Phi^M$: macroscopic specification, $\Phi^\mu$:
      a microscopic specification
Result: $\mathcal{T}$: a centralized LTS satisfying $\Phi^\mu$ and $\Phi^M$ and safe
        w.r.t. intermediate states
Translate the region graph into a specification $\Phi^{\mathbf{G_R}}$;
Create a specification $\Psi$ from $\Phi^M$ and $\Phi^\mu$;
Let $\Psi^{\text{safe}}$ be safety properties of $\Psi$;
Let $\Phi^C = \Phi^{\mathbf{G_R}} \wedge \Psi$;
Let $\Gamma = \Box(\texttt{True})$;
while $\Phi^C \wedge \Gamma$ is realizable do
    Synthesize a centralized LTS $\mathcal{T}$ satisfying $\Phi^C$;
    $\Gamma^{old} := \Gamma$;
    foreach transition $\tau = (q_i, q_{i+1}) \in \delta$ do
        $\phi_{q_i} := \bigwedge_{\pi_\mathbf{r} \in \mathcal{L}(q_i)} \pi_\mathbf{r} \wedge \bigwedge_{\pi_\mathbf{u} \in \Pi \setminus \mathcal{L}(q_i)} \neg \pi_\mathbf{u}$;
        $\phi_{q_{i+1}} := \bigwedge_{\pi_\mathbf{r} \in \mathcal{L}(q_{i+1})} \pi_\mathbf{r} \wedge \bigwedge_{\pi_\mathbf{u} \in \Pi \setminus \mathcal{L}(q_{i+1})} \neg \pi_\mathbf{u}$;
        $\phi_\tau := \bigwedge_{\pi_\mathbf{r} \in \mathcal{L}(q_i)} \phi_\mathbf{r} \wedge \bigwedge_{\pi_\mathbf{r} \in \beta} \neg \pi_\mathbf{r} \wedge \neg \phi_{q_i} \wedge \neg \phi_{q_{i+1}}$;
        if $\phi_\tau \not\models \Psi^{safe}$ then
            $\varphi := \Box(\phi_{q_i} \rightarrow \bigcirc(\neg \phi_{q_{i+1}}))$;
            $\Gamma := \Gamma \wedge \varphi$;
        end
    end
    if $\Gamma = \Gamma^{old}$ then
        return $\mathcal{T}$;
    end
end
return "No solution";
```

region A to C, then to E, then to C, then partitions the swarm such that a part moves to A and another part moves to B, then two parts gather in A, and this behavior is repeated. This LTS is also safe w.r.t. intermediate states.

4.1.4 Synthesis

Algorithm 1 summarizes the steps for computing a centralized LTS that satisfies the input specification. The specification $\Phi^C = \Phi^{\mathbf{G_R}} \wedge \Psi$ is the central specification that is used to synthesize a centralized LTS, obtained from the region graph and user provided specifications. The formula Γ, which is initially set to $\Box(\texttt{True})$, allowing all transitions, is updated at each iteration with a set of new formulas that rule out unsafe transitions. In the main loop of Algorithm 1 (lines 6–21) first a centralized LTS $\mathcal{T}$ is synthesized for $\Phi^C \wedge \Gamma$. Then each transition (q_i, q_{i+1}) of $\mathcal{T}$ is checked for safety. If a transition violates the specification, then a safety formula is added to Γ ruling out that transition. If after checking all transitions of $\mathcal{T}$, no new safety requirement is added ($\Gamma^{\text{old}} = \Gamma$), then $\mathcal{T}$ is safe and it can be returned as a solution. Otherwise, the process is repeated with the strengthened specification $\Phi^C \wedge \Gamma$.

Complexity The most costly step of Algorithm 1 is the synthesis of the centralized LTS $\mathcal{T}$ that satisfies $\Phi^C \wedge \Gamma$ (line 7). In Bloem et al. (2012), it is shown that synthesis for GR(1) specifications can be done with effort $O(mn|\Sigma_\mathcal{V}|^2)$ where m and n are the number of liveness assumptions and guarantees, respectively, and effort is measured in number of symbolic steps, i.e., image and pre-image computations. Let $\alpha = \alpha^M + \alpha^\mu$ be the number of liveness formulas in $\Phi^C \wedge \Gamma$ where α^M and α^μ are the number of liveness formulas in Φ^M and Φ^μ, respectively. Note that we do not have any liveness assumptions on the environment, nor do we add liveness formulas during synthesis. Hence, the synthesis of centralized LTS can be done in $O(\alpha|\Sigma_\mathcal{V}|^2)$ number of symbolic steps. At each iteration of Algorithm 1, at least one unsafe transition is ruled out. The number of possible transitions between the states is bounded by $O(|\Sigma_\mathcal{V}|^2)$. Thus, Algorithm 1 can compute a strategy with effort bounded by $O(\alpha|\Sigma_\mathcal{V}|^2.|\Sigma_\mathcal{V}|^2) = O(\alpha|\Sigma_\mathcal{V}|^4)$. Note that this is a crude upper bound, and in practice, Algorithm 1 may converge in much fewer iterations depending on the safety requirements and transition relation. Besides, at each iteration, all the transitions of the synthesized LTS are checked for safety, and possibly a *set* of unsafe transitions are ruled out.

Remark 2 An alternative method to synthesizing a centralized LTS is to explicitly model the intermediate states by adding additional propositions. To this end, we need to add an auxiliary proposition for each edge of the region graph, e.g., π_{AB} representing that a group is moving from region A to B, and their transition has not completed yet [similar to the approach proposed in Raman et al. (2015)]. This approach can synthesize an LTS in a single shot, however, it requires the addition of $|\mathbf{E}|$ propositions. Thus, the complexity is bounded by $O(\alpha(2^{|\mathbf{E}|}|\Sigma_\mathcal{V}|)^2)$ symbolic steps as the size of the state space grows exponentially with the number of propositions. Moreover, the symbolic steps themselves becomes more expensive to perform as the cost of symbolic computation strongly depends on the number of variables used in the Boolean formula (Bloem et al. 2006). In contrast, Algorithm 1 does not require auxiliary propositions to model the intermediate states. Note that not all the intermediate states need to be considered in the synthesis process, only those that appear in the solution.

4.2 Synthesis of decentralized controllers

Once a centralized LTS is obtained, the next step is to extract decentralized symbolic plans such that each can be assigned to a group of robots for execution. To this end, we form and solve an integer program that computes the minimum number of robots required to execute the centralized LTS, and shows how robots should move between regions to implement the transitions of the centralized LTS. The solution of the integer program is then used to partition the centralized LTS into a set of decentralized LTSs that each can be executed by a group consisting of one or more robots.

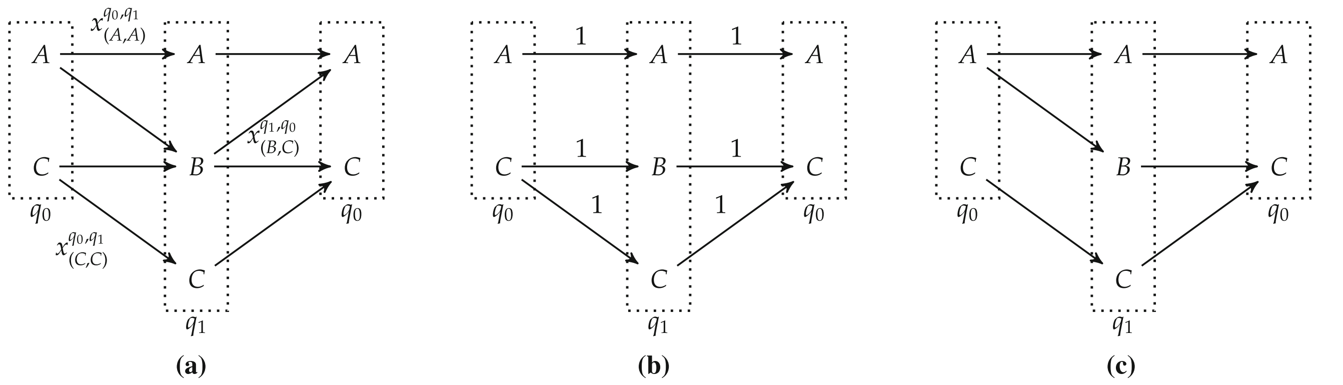

Fig. 4 **a** Possible transitions between regions. **b** A feasible matching. **c** An infeasible matching

4.2.1 Quantitative analysis of swarm size

To synthesize a centralized LTS $\mathcal{T} = \langle Q, q_0, \mathcal{V}, \delta, \mathcal{L}\rangle$ for the swarm, an abstraction is used that is independent of the number of robots in the swarm. Next we consider the problem of computing the minimum number of robots required to execute a symbolic plan. Let $\mathbf{R}_q \subseteq \mathbf{R}$ be the set of regions whose corresponding region proposition hold in $q \in Q$, i.e., $\mathbf{R}_q = \{\mathbf{r} \in \mathbf{R} \mid \pi_\mathbf{r} \in \mathcal{L}(q)\}$. We define $\mathbf{pre}(q) = \{u \in Q \mid (u, q) \in \delta\}$ and $\mathbf{post}(q) = \{v \in Q \mid (q, v) \in \delta\}$ to be (possibly singleton) sets of previous and next states for q in $\mathcal{T}$. For each transition $(q, v) \in \delta$, we define a set of non-negative integer variables $x^{qv}_{\mathbf{r}_q,\mathbf{r}_v} \in \mathbb{Z}_{\geq 0}$ where $\mathbf{r}_q \in \mathbf{R}_q$, $\mathbf{r}_v \in \mathbf{R}_v$, and $(\mathbf{r}_q, \mathbf{r}_v) \in \mathbf{E}$, i.e., there is an edge between $\mathbf{r}_q$ and $\mathbf{r}_v$ in the region graph.

Figure 4a shows possible transitions between regions for the LTS shown in Fig. 3b and based on the region graph shown in Fig. 3a. The middle dashed rectangular box correspond to state q_1, while the left and the right ones correspond to state q_0 in the LTS depicted in Fig. 3b (q_0 is duplicated to keep the figure simple). Some of the integer variables, e.g., $x^{q_0,q_1}_{(A,A)}$ are shown in Fig. 4a. The integer values assigned to these variables indicate how many robots must be sent from the source to the destination region, e.g., $x^{q_1,q_0}_{(B,C)} = 2$ means that two robots must move from region B to C during the transition from q_1 to q_0. Let $\mathcal{X}$ be the set of all integer variables. Naturally, it is desirable to minimize these quantities.

To this end, we define a set of linear constraints over these variables. First, the sum of the outgoing edges from a region must be greater than or equal to one, i.e.,

$$\sum_{(\mathbf{r}_q,\mathbf{r}_v)\in\mathbf{E}:\mathbf{r}_v\in\mathbf{R}_v} x^{qv}_{(\mathbf{r}_q,\mathbf{r}_v)} \geq 1 \qquad \forall (q, v) \in \delta \tag{2}$$

Furthermore, sum of the incoming edges to a region must be greater than or equal to one, i.e.,

$$\sum_{(\mathbf{r}_v,\mathbf{r}_q)\in\mathbf{E}:\mathbf{r}_q\in\mathbf{R}_q} x^{vq}_{(\mathbf{r}_v,\mathbf{r}_q)} \geq 1 \qquad \forall (v, q) \in \delta \tag{3}$$

Constraints 2 and 3 ensure that the regions corresponding to the region propositions that hold in the source and destination of a transition contain at least one robot. Note that a proposition $\pi_\mathbf{r}$ holds iff there is at least one robot in $\mathbf{r}$. Finally, for each state q, and for each state $u \in \mathbf{pre}(q)$ and $v \in \mathbf{post}(q)$, we define the *conservation* constraint:

$$\sum_{(\mathbf{r}_u,\mathbf{r}_q)\in\mathbf{E}:\mathbf{r}_u\in\mathbf{R}_u} x^{uq}_{(\mathbf{r}_u,\mathbf{r}_q)} = \sum_{(\mathbf{r}_q,\mathbf{r}_v)\in\mathbf{E}:\mathbf{r}_v\in\mathbf{R}_v} x^{qv}_{(\mathbf{r}_q,\mathbf{r}_v)} \tag{4}$$

i.e., sum of the robots entering a region is equivalent to the sum of the robots leaving that region or stay in it. Finally, we solve the integer program

$$\begin{aligned} &\min \sum_{x^{qv}_{\mathbf{r}_q,\mathbf{r}_v}\in\mathcal{X}} x^{qv}_{\mathbf{r}_q,\mathbf{r}_v} \\ &\text{subject to constraints (2), (3), and (4)} \end{aligned} \tag{5}$$

By solving (5), we obtain a vector $\mathbf{q} = (t^{\mathbf{r}_1}, \ldots, t^{\mathbf{r}_{|\mathbf{R}|}}) \in \mathbb{Z}^{|\mathbf{R}|}_{\geq 0}$ for each state $q \in Q$ characterizing the minimum number of robots required in each region in state q as follows: if $\mathbf{r}_j \in \mathcal{L}(q)$, then $t^{\mathbf{r}_j} = \sum_{(\mathbf{r}_j,\mathbf{r}_v)\in\mathbf{E}:\mathbf{r}_v\in\mathbf{R}_v} x^{qv}_{(\mathbf{r}_j,\mathbf{r}_v)}$ for an arbitrary chosen $v \in \mathbf{post}(q)$, and otherwise (i.e., $\mathbf{r}_j \notin \mathcal{L}(q)$), then $t^{\mathbf{r}_j} = 0$. Intuitively, if the region proposition does not hold, the corresponding vector element is zero, and otherwise it is the sum of the values attributed to the outgoing edges from that region to the regions of the next state in the LTS. The sum of the vector elements of $\mathbf{q}$, i.e., $\sum_{\mathbf{r}\in\mathbf{R}} t^\mathbf{r}$, is the minimum number of robots required to execute the computed LTS. Note that this sum is the same for any arbitrary $q \in Q$ due to the conservation constraint (4).

Example 7 Figure 4b shows a possible value assignment to integer variables corresponding to the edges in Fig. 4a. The edges whose corresponding variables are assigned to zero are not depicted. As shown in Fig. 4b, to implement the transition from q_0 to q_1, one robot stays in A, while from the two robots in C, one stays in C and the other one moves to B. Similarly,

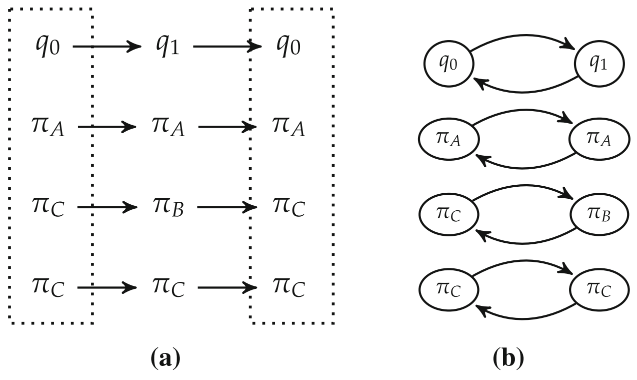

Fig. 5 **a** Partitioning of LTS in Fig. 3b, **b** three decentralized LTSs are obtained after partitioning

to implement the transition from q_1 to q_0, robots that are in A and C stay put, while the robot in region B moves to C. Observe that we need at least three robots to execute the symbolic plan shown in Fig. 3b. Intuitively, the solution of the integer program provide us with a "matching" between the regions, indicating how robots should move to implement transitions of the centralized LTS.

Example 8 A synthesized centralized LTS is not necessarily *feasible*, i.e., it cannot be implemented by a finite number of robots. As an example, assume that the edges (B, A) and (C, B) are removed from the region graph shown in Fig. 3a, i.e., the robots are not allowed to move from region B to A, or from C to B. Figure 4c shows possible transition between regions for the LTS shown in Fig. 3b consistent with the restricted region graph. Observe that although there is a matching between the regions, there is no value assignment to the edges that satisfies the conservation constraint in the corresponding integer program, i.e., the integer program is infeasible. Intuitively, every time the transition between q_0 to q_1 is executed, a robot must be "generated" to move from A to B, so we need an *infinite* number of robots to execute the symbolic plan.

4.2.2 Partitioning

The solution of the integer program (5) provides us with the minimum number of robots $k \geq 1$ required to implement the centralized LTS $\mathcal{T}$, and also *matchings* between the regions of the consecutive states of $\mathcal{T}$, i.e., how robots should move between regions to implement a transition of $\mathcal{T}$. The next step is to partition $\mathcal{T}$ to obtain k decentralized controllers. Each decentralized controller can be executed by a group of robots with a variable size. Partitioning of $\mathcal{T}$ is done by traversing its states one by one starting from its initial state; each state $q \in Q$ of $\mathcal{T}$ is partitioned and regions are assigned to k LTSs, according to the vector $\mathbf{q}$ corresponding to q and while maintaining consistency with the matchings obtained from the solution of (5), such that each state of the decentralized LTSs is labeled with a single region proposition.

Figure 5a shows the process of partitioning of the LTS shown in Fig. 3b. Each *stage* of partitioning contains the current state of the centralized LTS and labels of k decentralized LTSs, e.g., the left rectangular box in Fig. 5a depicts the stage $(q_0, \pi_A, \pi_C, \pi_C)$ where one decentralized LTS is labeled with π_A and the other two are labeled with π_C. Once a state of $\mathcal{T}$ is partitioned, we move to the next state according to the transition relation of $\mathcal{T}$. The second stage in Fig. 5a shows partitioning of q_1 and a matching that is consistent with Fig. 4b. This process of "unfolding" and partitioning the centralized LTS continues until a repeated stage is visited, e.g., the last stage in Fig. 5a is the same as initial stage. The repeated stage determines the looping index for decentralized LTSs. Figure 5b shows the three decentralized LTSs obtained through partitioning.

4.2.3 Synchronization skeletons for liveness

Figure 2b shows a partitioning of the LTS in Fig. 2a. After the robots visit region C for the second time (state q_3 of $\mathcal{T}$), the swarm is partitioned, one group moves to A and the other group moves to B. An initial synchronization skeleton for each decentralized controller is obtained while partitioning the centralized LTS. These initial synchronization skeletons indicate when each group must wait and synchronize with other groups to satisfy a liveness guarantee, e.g., to gather in some region. Formally, at each stage of partitioning $\left(q, \pi^1, \ldots, \pi^k\right)$, if the centralized LTS $\mathcal{T}$ synchronizes in q, i.e., $\pi_s \in \mathcal{L}(q)$, then $\mathcal{T}^i$ synchronizes with every other LTS $\mathcal{T}^j$, $j \neq i$ in the corresponding state. Intuitively, if the centralized LTS requires synchronization at a particular state, then all the decentralized controllers are set to synchronize with each other in the states corresponding to the partitioned state, e.g., $\mathcal{T}_1$ and $\mathcal{T}_2$ in Fig. 2b synchronize with each other at states u_0 and v_0, respectively.

The *synchronous* composition of LTSs $\mathcal{T}_1, \ldots, \mathcal{T}_k$ obtained after partitioning satisfies the given specification. However, this might not be true if the LTSs are executed *asynchronously*. Although the synchronization skeletons obtained in the previous step ensure that robots will synchronize to satisfy the liveness requirements, they may not be enough to guarantee safety, e.g., assume the LTSs in Fig. 2b are at states u_2 and v_2, respectively. If $\mathcal{T}_2$ moves to v_3 and then v_4, while $\mathcal{T}_1$ is still in u_2, then the joint state (u_2, v_4) is reached that violates the safety requirement $\Box(\neg(\pi_B \wedge \pi_E))$. Next, we show how we can reinforce the synchronization skeletons (if necessary) such that the safety is also guaranteed under asynchronous execution.

Remark 3 If the input specification does not have any safety requirement, i.e., $\Psi^{\text{safe}} = \texttt{True}$, then there is no need for reinforcing the synchronization skeletons. and the decentralized LTSs with the corresponding synchronization skeletons for liveness are returned as a solution.

4.3 Reinforcing the synchronization skeletons

Asynchronous composition of a set of transition systems captures all possible interleavings of their executions. If we have k transition systems each with n states, the composition can have n^k states, i.e., exponentially larger than the transition systems. However, for safety properties, the exact interleavings of the states does not matter, rather it is enough to know whether an unsafe state is reachable due to asynchronous execution of the LTSs or not, and in the former case to add extra synchronizations to make that unsafe state unreachable. We avoid explicit construction of the asynchronous composition of LTSs by taking advantage of this observation. To this end, we first introduce the notion of the *execution frame* for an LTS.

4.3.1 Execution frames

Intuitively, an execution frame (or frame for short) from an initial state is a sequence of states that an LTS can visit during its execution until either reaching a synchronization state or a state that was already visited. Formally, a frame $f = \langle q_{i_0}, q_{i_1}, \ldots, q_{i_j} \rangle$ from an initial state q_{i_0} is a maximal sequence of states such that $(q_{i_p}, q_{i_{p+1}}) \in \delta$ for all $0 \leq p < j$, and either $\exists 0 \leq p < j.\ q_{i_p} = q_{i_j}$ and $\forall p, r.\ 0 \leq p < r < j : q_{i_p} \neq q_{i_r}$ (i.e., q_{i_j} is the first repeated state), or $\pi_s \in \mathcal{L}(q_{i_j})$ and $\forall 0 < p < j.\ \pi_s \notin \mathcal{L}(q_{i_p})$ (i.e., q_{i_j} is the first state after q_{i_0} that synchronization is required). For example, LTSs $\mathcal{T}_1$ and $\mathcal{T}_2$ in Fig. 2b have only one execution frame $\langle u_0, u_1, \ldots, u_4, u_0 \rangle$ and $\langle v_0, v_1, \ldots, v_4, v_0 \rangle$, respectively.

At any time during execution, each LTS is in one of its frames. For a frame $f = \langle q_i, q_{i+1} \cdots q_j \rangle$, let $[\![f]\!] = \{q_i, q_{i+1}, \ldots, q_j\}$ be the set of states appearing in f. A *frame profile* $\mathcal{FP} = (f_1, \ldots, f_k)$ for a set of LTSs $\mathcal{T}_1, \ldots, \mathcal{T}_k$ is a tuple where each f_i is a possible frame for $\mathcal{T}_i$. Note that due to synchronizations between LTSs, not all frame profiles are reachable in general. Only reachable frame profiles are considered while reinforcing the synchronization skeletons.

4.3.2 Checking the safety of the frame profiles

Consider the LTSs in Fig. 2b executing their only frames. The processes may proceed with different speeds. Indeed there are 252 different interleavings that take the system from joint state (u_0, v_0) back to (u_0, v_0). Instead of constructing these different interleavings, we encode the set of reachable states *symbolically* as $\phi = (u_0 \vee u_1 \vee \cdots \vee u_4) \wedge (v_0 \vee v_1 \vee \cdots \vee v_4)$. Intuitively, ϕ specifies that any combination of states of LTSs are reachable, i.e., $\mathcal{T}_1$ is in one of the states $\{u_0, \ldots, u_4\}$ and $\mathcal{T}_2$ is in one of the states $\{v_0, \ldots, v_4\}$. Formally, given a frame profile $\mathcal{FP} = (f_1, \ldots, f_k)$ for LTSs $\mathcal{T}_1, \ldots, \mathcal{T}_k$, we obtain a predicate $\phi_{\mathcal{FP}} = \bigwedge_{i=1}^{k} (\bigvee_{q \in [\![f_i]\!]} q)$ that encodes the set of all reachable states due to the different interleavings.

Next, we check if there are any unsafe states for a frame profile $\mathcal{FP}$. To this end, the conjunction of the frame profile formula $\phi_{\mathcal{FP}}$ and negation of the safety properties of the specification is computed. If the conjunction is unsatisfiable [this can be checked using SAT solvers (Claessen et al. 2008) or BDDs (Bryant 1992)], then $\mathcal{FP}$ is safe, i.e., no unsafe state is reachable in $\mathcal{FP}$. Otherwise, $\mathcal{FP}$ is unsafe, e.g., the joint state (u_2, v_4) is an unsafe state for the LTSs in Fig. 2b, and additional synchronization is necessary to prevent reaching unsafe states.

4.3.3 Adding synchronizations for safety

Let n be the number of states for each LTS and l be the index of the state where the loop begins, e.g., for the LTSs in Fig. 2b we have $n = 5$ and $l = 0$. Assume $q = (q^1_{i_1}, q^2_{i_2}, \ldots, q^k_{i_k})$ is an unsafe joint state for LTSs $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$, where $0 \leq i_j < n$ for $1 \leq j \leq k$. Let $\mathbf{min}_q = min(i_1, \ldots, i_k)$ and $\mathbf{max}_q = max(i_1, \ldots, i_k)$ be the minimum and maximum of the indices in the unsafe joint state q. Note that since the synchronous execution of the decentralized LTSs satisfies the specification and is safe w.r.t. intermediate states (due to the construction in Algorithm 1), any two joint states $q_i = (q^1_i, \ldots, q^k_i)$ and $q_j = (q^1_j, \ldots, q^k_j)$ such that either $j = i + 1$, or $j = n - 1$ and $i = l$ (i.e., the joint states corresponding to consecutive states in the centralized LTS) are safe. In addition, all possible intermediate states between q_i and q_j are also safe. Thus, for any unsafe joint state q, we have $(\mathbf{max}_q \geq \mathbf{min}_q + 2)$.

Our goal is to make the unsafe states unreachable by adding synchronizations. Note that to reach the unsafe joint state q, each LTS $\mathcal{T}_j$ must reach its corresponding state $q^j_{i_j}$. Intuitively, an LTS $\mathcal{T}_j$ may reach the state $q^j_{i_j}$ in two ways; either by starting from a state q^j_ℓ with $\ell \leq i_j$ and proceeding such that the indices increase along the path to $q^j_{i_j}$, or by starting from a state q^j_ℓ with $\ell > i_j$ and reaching q_{i_j} after "looping", i.e., after visiting the state q^j_l. The latter is only possible if $i_j \geq l$. To ensure that the unsafe state q is unreachable, we make sure that the LTSs synchronize in at least one state with index $\mathbf{min}_q < i < \mathbf{max}_q$. Furthermore, if $\mathbf{min}_q \geq l$, i.e., the minimum index corresponding to the unsafe state q is greater than or equal to the looping index, we add at least one synchronization after the index $\mathbf{max}_q$ and before the index $\mathbf{min}_q$ so that the unsafe state cannot be reached due to the looping. Formally, if $\mathbf{min}_q \geq l$, there must be a synchronization at an index $i \in [l, \mathbf{min}_q) \cup (\mathbf{max}_q, n - 1]$.

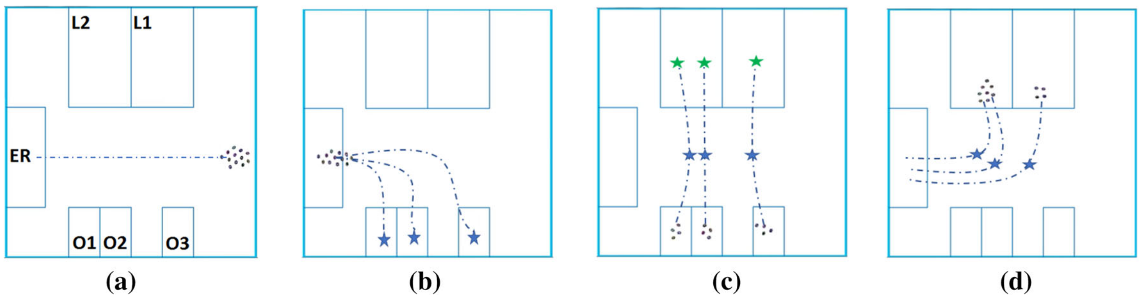

Fig. 6 Three groups of robots navigating between labs, offices and an equipment room through a corridor

We enumerate the unsafe states for every reachable frame profile, and obtain a set of intervals as potential candidates for synchronization to ensure safety. We then compute a minimal set of indices that covers all intervals,[1] thus making all unsafe states unreachable. These additional synchronizations combined with synchronization skeletons computed for liveness, form the final synchronization skeletons that ensure satisfaction of the liveness and safety properties.

Example 9 To ensure that the swarm never occupies regions E and B at the same time, i.e., the unsafe joint state (u_2, v_4) cannot be reached, $\mathcal{T}_1$ and $\mathcal{T}_2$ synchronize at the states u_3 and v_3 in Fig. 2b (note that $3 \in (2, 4)$). To ensure that (u_2, v_4) cannot be reached due to looping, there must be at least a synchronization at an index chosen from $[0, 2) \cup (4, 4]$, i.e., at states u_0 and v_0, or u_1 and v_1. Since $\mathcal{T}_1$ and $\mathcal{T}_2$ already synchronize in v_0 and u_0 for liveness, the same synchronization can also be used to prevent reaching an unsafe state.

4.4 Synthesis of decentralized controllers

Algorithm 2 summarizes the steps to synthesize decentralized LTSs from high-level task specifications. Note that computed LTSs may include redundant "stuttering" states, i.e., states where the robots stay in the same region without synchronizing with any other group. As a final step, these states are removed from the solutions to obtain shorter LTSs, e.g., $\mathcal{T}_1$ in Fig. 2b is shortened by removing the state u_4.

Algorithm 2 synthesizes provably-correct symbolic plans that realize the given specification. Continuous controllers that *simulate* the symbolic plans can then be obtained using existing methods in the literature (see e.g., Tabuada 2009). If such continuous controllers exist that faithfully implement the symbolic plans, then the correct system behavior is guaranteed at the continuous level, as we have recently shown (Chen et al. 2018).

[1] This is a special case of the minimal hitting set problem (Kleinberg and Tardos 2006).

Algorithm 2: Synthesis of Decentralized Controllers

```
Data: Region graph $\mathbf{G_R}$, Macroscopic $\Phi^M$ and Microscopic $\Phi^\mu$
      specifications
Result: A set of LTSs $\mathcal{T}_1, \cdots, \mathcal{T}_k$ along with their
        synchronization skeletons $\mathcal{S}_1, \cdots, \mathcal{S}_k$
1 Synthesize a centralized LTS $\mathcal{T}$ that realizes $\Phi^M$ and $\Phi^\mu$, and is
  safe w.r.t. intermediate states;
2 Partition $\mathcal{T}$ into decentralized LTSs $\mathcal{T}_1, \cdots, \mathcal{T}_k$;
3 Extract initial synchronization skeletons $\mathcal{S}_1, \cdots, \mathcal{S}_k$;
4 Reinforce synchronization skeletons for safety;
5 Remove stuttering states;
```

5 Example

Consider a workspace partitioned into two labs $(L1, L2)$, three offices $(O1, O2, O3)$, and an equipment room (ER) that are connected to each other through a corridor C as shown in Fig. 6a. Let $\mathbf{R} = \{O1, O2, O3, ER, C, L1, L2\}$ be the set of all regions. All the robots, initially positioned at the right side of the corridor, must repeatedly visit the equipment room (possibly at different times). This is specified as the microscopic specification $\Phi^\mu = \forall \mathbf{a}.\ \Box\Diamond(\pi^{\mathbf{a}}_{ER})$. Moreover, the swarm must repeatedly occupy the three offices at the same time, described as

$$\Phi_1 = \Box\Diamond\left(\pi_{O1} \wedge \pi_{O2} \wedge \pi_{O3} \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{O1,O2,O3\}} \neg\pi_{\mathbf{r}}\right)$$

Similarly, two labs must be repeatedly occupied at the same time, encoded as

$$\Phi_2 = \Box\Diamond\left(\pi_{L1} \wedge \pi_{L2} \wedge \bigwedge_{\mathbf{r}\in\mathbf{R}\setminus\{L1,L2\}} \neg\pi_{\mathbf{r}}\right)\Bigg)$$

Finally, if a part of the swarm is present in the labs, there must be no robots in the offices and vice versa. This is captured by

$$\Phi_3 = \Box(\neg((\pi_{L1} \vee \pi_{L2}) \wedge (\pi_{O1} \vee \pi_{O2} \vee \pi_{O3})))$$

The macroscopic specification is given as $\Phi^M = \bigwedge_{i=1}^{3} \Phi_i$.

We implemented our algorithms in Java. The simulations were done in the Robotarium simulator in Matlab (Pickem et al. 2016). A video of a simulation accompanies this paper The demonstrations were run on a desktop machine with an Intel Core i7 CPU@3.40GHz and 16GB RAM. The centralized LTS was computed in 11 ms. Reinforcing the synchronization skeletons took 47 ms.

Three LTSs along with their corresponding synchronization skeletons were synthesized and assigned to three groups of robots each with four members. The synthesized centralized LTS and each of the three resulting decentralized LTSs have 6 states (not considering the intermediate states). Figure 6a–d show how three groups navigate through the workspace. The dashed lines show the general directions each group moves in and the stars represent regions where synchronizations happen between groups. The synthesized LTSs guide the robots from their initial position toward the equipment room. Then, each LTS guide the robots to one of the offices. Once all three groups entered their corresponding offices, they synchronize and then two groups move to $L2$ and one group moves to $L1$. Groups gather in the corridor after visiting the labs and synchronize, and then they repeat visiting the equipment room, offices and labs as before.

6 Experimental results

In this section we study how varying the complexity of the input specification can affect the computation time for synthesis of centralized LTSs and synchronization skeletons. We consider a $N \times M$ grid-world partitioned into cells (regions) $\mathbf{R} = \{\mathbf{r}_{i,j} \mid 1 \leq i \leq N \text{ and } 1 \leq j \leq M\}$. We assume that a swarm of robots is initially positioned in the top-left cell $\mathbf{r}_{1,1}$ which is a charging station. The robots need to repeatedly visit the charging station, but they can do so at different times. This requirement is modeled as a liveness formula in the specified microscopic specification, i.e., $\Phi^{\mu} = \forall \mathbf{a}.\ \Box\Diamond \mathbf{r}^{\mathbf{a}}_{1,1}$. In the sequel, we vary the size of the grid-world, the number of liveness and safety requirements, and number of regions that must be occupied at the same time by the swarm, and report and discuss our experimental results.

Increasing the number of regions To study the effects of the number of regions on the synthesis time, we fixed the input specification Φ as follows: (i) microscopic specification Φ^{μ}: all the robots in the swarm must repeatedly visit the charging station, (ii) macroscopic specification Φ^{M}: the swarm must repeatedly occupy the top-right and bottom-left cells at the same time, i.e., $\Phi^{M} = \Box\Diamond(\mathbf{r}_{1,M} \wedge \mathbf{r}_{N,1} \wedge \bigwedge_{\mathbf{r} \in \mathbf{R} \setminus \{\mathbf{r}_{1,M}, \mathbf{r}_{N,1}\}} \neg \mathbf{r})$. Figure 7 shows the computation times of the centralized LTSs for different sizes of the grid-world. As can be seen there, the synthesis time grows exponentially with the number of regions.

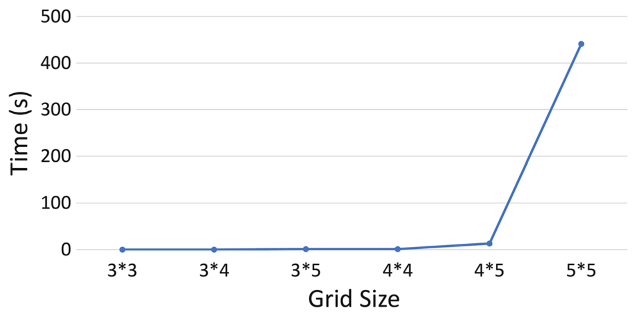

Fig. 7 Computation time for synthesis of centralized LTSs for different grid-worlds sizes

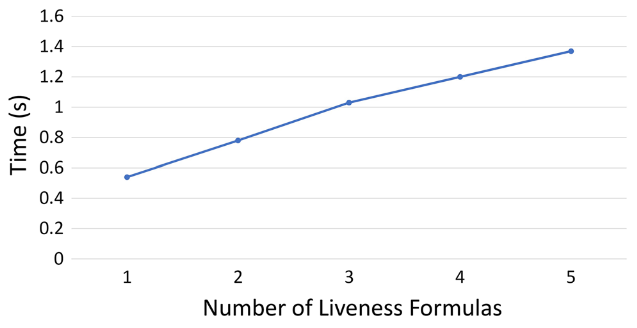

Fig. 8 Computation time for synthesis of centralized LTSs for a 4×4 grid-world w.r.t. different number of liveness formulas

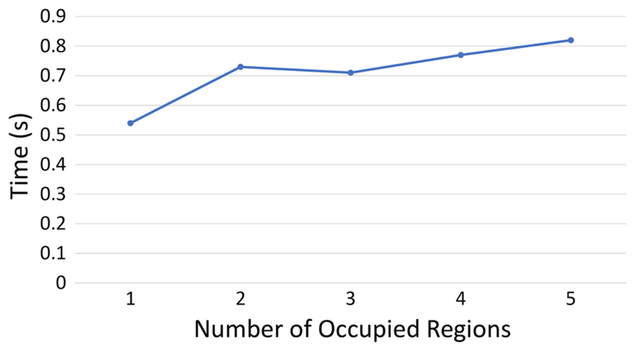

Fig. 9 Computation time for synthesis of centralized LTSs for a 4×4 grid-world where the swarm must occupy different number of regions at the same time

Increasing the number of liveness formulas Next, we consider a 4×4 grid-world. The microscopic specification is defined as before. For the macroscopic specification, we generate β number of liveness formulas $\Phi_1, \Phi_2, \ldots, \Phi_\beta$ where $\Phi_k = \Box\Diamond(\mathbf{r}_{i,j} \bigwedge_{\mathbf{r} \in \mathbf{R} \setminus \{\mathbf{r}_{i,j}\}} \neg \mathbf{r})$ for $1 \leq k \leq \beta$, i.e., each liveness formula Φ_k requires the whole swarm to occupy a region $\mathbf{r}_{i,j}$ with $\mathbf{r}_{i,j}$ being selected randomly. Figure 8 shows the synthesis time w.r.t. the number of liveness formulas β. For each value of β, we ran the experiment five times and reported the average synthesis time. As it can be seen in Fig. 8, the synthesis time grows linearly with the number of liveness formulas.

Table 1 Evaluation of effects of safety formulas on computation time of centralized LTS and synchronization skeleton

Size	# Safety	# Occ. regions	$\|\mathcal{T}\|$		Synthesis time (s)			Iterations			Safety synch. time (ms)		
			Min	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
4×4	0	3	12	12	0.71	0.72	0.75	1	1	1	7	9.2	15
4×4	1	3	12	12	0.7	1.9	2.9	1	1.8	3	8	40.4	153
4×4	2	3	12	12	0.6	5.6	22.7	1	4.6	19	8	56	119
4×4	3	3	12	12	2.8	4	6.5	1	2.8	5	23	51	76
4×4	4	3	12	12	1.3	11.3	25.9	1	4.75	8	49	70.5	93
4×4	5	3	12	12	1.7	17.3	78	1	6.25	22	54	145.25	326
4×4	0	2	6	6	0.58	0.61	0.64	1	1	1	2	2.25	3
4×4	1	2	6	8	0.6	1.08	1.4	1	1.25	2	2	3.25	7
4×4	2	2	6	8	0.8	1.76	3.6	1	1.2	2	2	3.8	7
4×4	3	2	6	10	0.6	3.92	7.8	1	2.8	9	2	3.6	5
4×4	4	2	6	10	1.7	2.28	2.9	1	1.2	2	2	3	5
4×4	5	2	6	10	1.2	6.74	24	1	3.2	12	2	2.8	4
3×4	5	2	6	10	0.17	0.33	0.9	1	5.2	21	2	3	4
3×4	5	3	10	10	0.14	1.25	2.9	1	13	28	21	115.6	271

The columns show the size of the grid-world, number of safety formulas, number of regions that must be occupied at the same time in the liveness formula of the macroscopic specification, the minimum and maximum lengths of computed centralized LTSs, the minimum, average, and maximum synthesis time, number of iterations required to compute the centralized LTS, and the time needed to reinforce the synchronization skeleton, respectively

Increasing the number of simultaneously occupied regions Here, we consider a 4×4 grid-world, and we assume that the microscopic specification is defined as before. The macroscopic specification is defined as a single liveness formula that requires the swarm to occupy γ randomly-selected regions at the same time, i.e., $\Phi^M = \Box\Diamond((\bigwedge_{\mathbf{r}\in\mathbf{R}^\gamma} \mathbf{r}) \wedge (\bigwedge_{\mathbf{u}\in\mathbf{R}\setminus\mathbf{R}^\gamma} \neg\mathbf{u}))$ where $\mathbf{R}^\gamma \subseteq \mathbf{R}$ is a set of γ randomly-selected regions. Figure 9 shows how the synthesis time changes as we increase the number of the regions that the swarm must occupy at the same time. As it can be seen in Fig. 9, the synthesis time slightly grows while increasing γ. Also, partitioning of the centralized LTS lead to γ number of decentralized LTSs since the swarm needs to occupy γ regions at the same time. The time required to partition the centralized LTS was insignificant (less than one millisecond) in all experiments.

Safety Next, we study how changing the number of safety requirements will affect the synthesis time, number of iterations required to synthesize a centralized LTS, and the time needed to reinforce the synchronization skeletons. To this end, in each experiment, we fix the size of the grid-world. The microscopic specification requires all the robots to repeatedly visit the charging station. The macroscopic specification includes one liveness formula that requires the whole swarm to repeatedly visit either two regions (top-right and bottom-left cells), or three regions (top-right, bottom-left, and bottom-right cells) at the same time (see # *Occ. Regions* column in Table 1). Furthermore, the macroscopic specification includes θ number of safety formulas where each safety formula is of the form $\Phi_s = \Box(\neg(\mathbf{r}_{i_1,j_1} \wedge \mathbf{r}_{i_2,j_2}))$ with $\mathbf{r}_{i_1,j_1}$ and $\mathbf{r}_{i_2,j_2}$ being two different randomly-selected regions. Intuitively, each safety formula requires the swarm not to occupy two specified regions at the same time.

For each grid-world size and input specification, we repeated the experiment five times, and report the average, min and max computation times as shown in Table 1. The columns show the size of the grid-world, number of safety formulas, number of regions that must be occupied at the same time in the liveness formula of the macroscopic specification, the minimum and maximum lengths of computed centralized LTSs, the minimum, average, and maximum synthesis time, number of iterations required to compute the centralized LTS, and the time needed to reinforce the synchronization skeleton, respectively. The time required to partition the centralized LTS was insignificant (less than one millisecond) in all experiments.

As it can be observed in Table 1, the synthesis time does not directly depend on the number of safety formulas. Since the synthesis is done symbolically, the complexity of the underlying symbolic data structures (BDDs) that represent the safety requirements can affect the synthesis time. Furthermore, the synthesis time also depends on how "restrictive" the safety requirements are, e.g., if the initial synthesized LTS is safe w.r.t. safety requirements, the synthesis is done in one iteration, otherwise, the unsafe transitions are ruled out in possibly several iterations, leading to increased computation time. For example, in our experiments with a 4×4

grid-world with 5 safety formulas where three regions must be repeatedly visited at the same time, in one of the experiments, synthesis of centralized LTS was done in 1.7 s in one iteration, while in another experiment, synthesis was done in 78 s and in 22 iterations (see Table 1).

The time needed to reinforce the synchronization skeletons for safety depends on the safety requirements, the length and number of the decentralized LTSs. In the experiments shown in Table 1, the number of decentralized LTSs are the same as the number of regions that must be occupied at the same time. The average time required to reinforce the synchronization skeletons increase as the length and number of decentralized LTSs increase, as it can be seen in Table 1. Finally, we observe that the bottleneck of the framework seems to be the synthesis of the centralized LTS. To improve the scalability of the proposed framework, the first step would be to boost the scalability of centralized synthesis. To this end, compositional synthesis algorithms (Alur et al. 2018; Filiot et al. 2011) seem like a promising future direction.

7 Conclusions and future work

We presented a framework for high-level specification of swarm behaviors and automated synthesis of decentralized controllers and synchronization skeletons. In this paper, we assumed that the environment is static and known, i.e., it is not changing and the values of all the propositions are controlled by the system. In general, GR(1) allows one to specify and synthesize controllers that *react* to a changing environment. Roughly speaking, in the presence of a dynamically-changing and possibly adversarial environment, symbolic plans are no longer a sequence of steps, but *strategies* that capture how the system must react to possible changes in its environment. Some of the most challenging aspects of reactive controller synthesis for swarms is to consider that the environment can change asynchronously and even during the intermediate states. Therefore, the swarm must be prepared to react in all possible scenarios. Future work includes extending the work in this paper to tasks performed in dynamically changing environments.

Acknowledgements The authors thank Rüdiger Ehlers for insightful discussions relating to the need for quantitative analysis of the synthesized symbolic plan.

References

Alur, R., Moarref, S., & Topcu, U. (2018). Compositional and symbolic synthesis of reactive controllers for multi-agent systems. *Information and Computation*, *261*, 616–633.

Bachrach, J., Beal, J., & McLurkin, J. (2010). Composable continuous-space programs for robotic swarms. *Neural Computing and Applications*, *19*(6), 825–847.

Balch, T., & Hybinette, M. (2000). Social potentials for scalable multi-robot formations. In *IEEE international conference on robotics and automation, 2000. Proceedings. ICRA'00* (Vol. 1, pp. 73–80). IEEE.

Bloem, R., Gabow, H. N., & Somenzi, F. (2006). An algorithm for strongly connected component analysis in n log n symbolic steps. *Formal Methods in System Design*, *28*(1), 37–56.

Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., & Sa'ar, Y. (2012). Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, *78*(3), 911–938.

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1–41.

Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, *24*(3), 293–318.

Chen, J., Moarref, S., & Kress-Gazit, H. (2018). Verifiable control of robotic swarm from high-level specifications. In *Proceedings of the 17th international conference on autonomous agents and multiagent systems (AAMAS 2018)* (to appear).

Church, A. (1962). Logic, arithmetic and automata. In *Proceedings of the international congress of mathematicians* (pp. 23–35).

Claessen, K., Een, N., Sheeran, M., & Sorensson, N. (2008). Sat-solving in practice. In *9th international workshop on discrete event systems, 2008. WODES 2008* (pp. 61–67). IEEE.

Clarke, E. M., Emerson, E. A., & Sifakis, J. (2009). Model checking: Algorithmic verification and debugging. *Communications of the ACM*, *52*(11), 74–84.

Dixon, C., Winfield, A. F., Fisher, M., & Zeng, C. (2012). Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems*, *60*(11), 1429–1441.

Ehlers, R. (2010). Symbolic bounded synthesis. In *International conference on computer aided verification* (pp. 365–379). Berlin: Springer.

Emerson, E. A., & Clarke, E. M. (1982). Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, *2*(3), 241–266.

Filiot, E., Jin, N., & Raskin, J. F. (2011). Antichains and compositional algorithms for LTL synthesis. *Formal Methods in System Design*, *39*(3), 261–296.

Filippidis, I., Dimarogonas, D. V., & Kyriakopoulos, K. J. (2012). Decentralized multi-agent control from local LTL specifications. In *2012 IEEE 51st annual conference on decision and control (CDC)* (pp. 6235–6240). IEEE.

Gjondrekaj, E., Loreti, M., Pugliese, R., Tiezzi, F., Pinciroli, C., Brambilla, M., Birattari, M., & Dorigo, M. (2012). Towards a formal verification methodology for collective robotic systems. In *Formal methods and software engineering* (pp. 54–70). Berlin: Springer.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading: Addison-Wesley.

Guo, M., & Dimarogonas, D. V. (2015). Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research*, *34*(2), 218–235.

Guo, M., Tumova, J., & Dimarogonas, D. V. (2014). Cooperative decentralized multi-agent control under local LTL tasks and connectivity constraints. In *2014 IEEE 53rd annual conference on decision and control (CDC)* (pp. 75–80). IEEE.

Karaman, S., & Frazzoli, E. (2008). Vehicle routing with linear temporal logic specifications: Applications to multi-UAV mission planning. In *AIAA guidance, navigation and control conference and exhibit* (p. 6838).

Kleinberg, J., & Tardos, E. (2006). *Algorithm design*. London: Pearson Education India.

Kloetzer, M., & Belta, C. (2006). Hierarchical abstractions for robotic swarms. In *Proceedings 2006 IEEE international conference on robotics and automation, 2006. ICRA 2006* (pp. 952–957). IEEE.

Kloetzer, M., Ding, X. C., & Belta, C. (2011). Multi-robot deployment from LTL specifications with reduced communication. In *2011 50th IEEE conference on decision and control and European control conference (CDC-ECC)* (pp. 4867–4872). IEEE.

Labella, T. H., Dorigo, M., & Deneubourg, J. L. (2006). Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, *1*(1), 4–25.

Loizou, S. G., & Kyriakopoulos, K. J. (2005). Automated planning of motion tasks for multi-robot systems. In *44th IEEE conference on decision and control, 2005 and 2005 European control conference. CDC-ECC'05* (pp. 78–83). IEEE.

Manna, Z., & Wolper, P. (1984). Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *6*(1), 68–93.

Moarref, S., & Kress-Gazit, H. (2017). Decentralized control of robotic swarms from high-level temporal logic specifications. In *2017 international symposium on multi-robot and multi-agent systems (MRS)* (pp. 17–23). IEEE.

Nilsson, P., & Ozay, N. (2016). Control synthesis for large collections of systems with mode-counting constraints. In *Proceedings of the 19th international conference on hybrid systems: Computation and control* (pp. 205–214). ACM.

Nouyan, S., Campo, A., & Dorigo, M. (2008). Path formation in a robot swarm. *Swarm Intelligence*, *2*(1), 1–23.

Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-agent Systems*, *11*(3), 387–434.

Pickem, D., Glotfelter, P., Wang, L., Mote, M., Ames, A., Feron, E., & Egerstedt, M. (2016). The robotarium: A remotely accessible swarm robotics research testbed. arXiv preprint arXiv:1609.04730.

Pnueli, A., & Rosner, R. (1989). On the synthesis of a reactive module. In *Proceedings of the 16th ACM symposium on principles of programming languages* (pp. 179–190). ACM.

Raman, V., & Kress-Gazit, H. (2014). Synthesis for multi-robot controllers with interleaved motion. In *2014 IEEE international conference on robotics and automation (ICRA)* (pp. 4316–4321). IEEE.

Raman, V., Piterman, N., Finucane, C., & Kress-Gazit, H. (2015). Timing semantics for abstraction and execution of synthesized high-level robot control. *IEEE Transactions on Robotics*, *31*(3), 591–604.

Rosner, R. (1992). *Modular synthesis of reactive systems*. Ph.D. thesis, Weizmann Institute of Science.

Soysal, O., & Sahin, E. (2005). Probabilistic aggregation strategies in swarm robotic systems. In *Swarm intelligence symposium, 2005. SIS 2005. Proceedings 2005 IEEE* (pp. 325–332). IEEE.

Tabuada, P. (2009). *Verification and control of hybrid systems: A symbolic approach*. Berlin: Springer.

Ulusoy, A., Smith, S. L., Ding, X. C., Belta, C., & Rus, D. (2013). Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, *32*(8), 889–911.

Vechev, M., Yahav, E., & Yorsh, G. (2010). Abstraction-guided synthesis of synchronization. In *ACM Sigplan notices* (Vol. 45, pp. 327–338). ACM.

Salar Moarref is a postdoctoral associate at the Sibley School of Mechanical and Aerospace Engineering at Cornell University. He received his Ph.D. in Computer and Information Science from the University of Pennsylvania in 2016. His research focuses on developing algorithms and tools for automated synthesis of controllers for multi-robot and multi-agent systems from high-level specifications.

Hadas Kress-Gazit is an Associate Professor at the Sibley School of Mechanical and Aerospace Engineering at Cornell University. She received her Ph.D. in Electrical and Systems Engineering from the University of Pennsylvania in 2008 and has been at Cornell since 2009. Her research focuses on formal methods for robotics and automation and more specifically on creating verifiable robot controllers for complex high-level tasks using logic, verification, synthesis, hybrid systems theory and computational linguistics. She received an NSF CAREER award in 2010, a DARPA Young Faculty Award in 2012 and the Fiona Ip Li '78 and Donald Li '75 Excellence in teaching award in 2013.