2008 IEEE International Conference on
Robotics and Automation
Pasadena, CA, USA, May 19-23, 2008

Distributed implementations of global temporal logic motion specifications

Marius Kloetzer and Calin Belta
Center for Information and Systems Engineering
Boston University
15 Saint Mary's Street, Boston, MA 02446
{kmarius,cbelta}@bu.edu

Abstract— **We present a computational framework for automatic synthesis of decentralized communication and control strategies for a robotic team from global specifications given as temporal and logic statements about visiting regions of interest in a partitioned environment. We consider a purely discrete scenario where the robots move among the vertices of a graph. However, by employing recent results on invariance and facet reachability for dynamical system in environments with polyhedral partitions, the framework from this paper can be directly implemented for robots with nontrivial dynamics. While providing a rich specification language and guaranteeing the correctness of the solution, our approach is conservative, in the sense that we might not find a solution even if one exists. The overall amount of required computation is large. However, most of it is performed off-line before the deployment.**

I. Introduction

Planning and controlling robot motion is a challenging problem that received a lot of attention in recent years [1]. The goal is to be able to specify a task in a rich, high level language and have the robot(s) automatically convert this specification into a set of low level primitives, such as feedback controllers and communication protocols, to accomplish the task.

This work is motivated by two disadvantages of the current approaches to robot motion planning and control. First, in most of the existing works, the motion planning problem is simply specified as "go from *A* to *B*" [1], and this is not rich enough to describe a large class of tasks of interest in practical applications. Second, most current approaches to multi-robot planning and control are bottom-up, in the sense that local interaction rules, thought to be true in natural systems [2] (*e.g.,* schools of fish, swarms of bees) or designed from first principles [3], are shown to produce emergent behavior at global level (largely known as "consensus algorithms"). However, the inverse (top-down) problem, which seems more relevant to robotics, remains largely unanswered: "Can we automatically generate provably correct local communication and control strategies from task specifications given in rich and natural language over regions of interest in an environment?"

The starting point for this paper is the observation that "rich" and "human-like" task specifications, such as the temporal and logic statements about the reachability of regions of interest given above, translate naturally to formulas of temporal logics, such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) [4]. Such logics and corresponding model checking algorithms are normally used to specify and check the correctness of computer programs, which can be seen as continuously operating, reactive (concurrent) systems [5].

Inspired by this, here we consider a purely discrete problem, in which n agents can move among a finite number of locations. We consider $n+1$ relations over the set of locations. One relation captures communication constraints among the agents as a function of their position. Each of the remaining n relations captures each agent's motion constraints. We present a framework for automatic generation of local control strategies from a task specification given as an arbitrary LTL formula over the set of locations.

It is important to note that, due to recent results on construction of feedback controllers for facet reachability and invariance in polytopes [6]–[8], the solution to this purely discrete and finite dimensional problem can be actually implemented by robots with non-trivial dynamics moving in environments partitioned using popular schemes such as triangulations and rectangular grids [9].

The use of temporal logic for task specification and controller synthesis in mobile robotics has been advocated as far back as [10], and recent works include [11]–[13] and our own work [14]. As opposed to [11], [13], [14], here we consider teams, rather than just single agents. As opposed to [12], we focus on synthesis of provably correct strategies, rather than on verification of existing ones. As far as we know, this is the first attempt to constructing a framework for automatic construction of distributed control strategies from global specifications given as temporal and logic statements over a partitioned environment. Arguably, the closest related works are recent results in concurrency theory [15], where global specifications given as languages over a set of actions are checked for implementability by a set of agents jointly owning the actions and synchronizing on them. However, in the robotics problem that we consider, there is no easy way to define this set of actions. Consequently, we chose to start by constructing a "rich" centralized solution and then pruning it to guarantee distributed implementability. In this regard, this paper can be seen as an extension of our previous work [16], where the communication architecture was centralized.

This work was partially supported by NSF CAREER 0447721 and NSF 0410514 at Boston University.

M. Kloetzer is the corresponding author.

II. Problem Formulation

We consider a discrete environment with a finite set of locations $P = \{p_1, \ldots, p_k\}$, and a team of n, $n < k$, agents that can move in this environment. The motion capabilities of each agent i, $i = 1, \ldots, n$, are captured by a relation:

$$\alpha_i \subseteq P \times P, \tag{1}$$

in the sense that robot i can move from p_i to p_j while not visiting other locations if and only if $(p_i, p_j) \in \alpha_i$.

The agents are assumed to have identical communication capabilities, which can be modelled as the following reflexive and symmetric relation:

$$\sim \subseteq P \times P, \tag{2}$$

where $(p_i, p_j) \in \sim$ if and only if an agent located in p_i can directly communicate with an agent located in p_j.

We make the following **assumptions**: (1) Each agent can stay in its current location (*i.e.,* $(p_j, p_j) \in \alpha_i$, for any $i = 1, \ldots, n$ and $j = 1, \ldots, k$); (2) Initially, the agents are in different locations, (3) One agent can move only between locations from which robots can communicate (*i.e.,* $\alpha_i \subseteq \sim$); (4) Each agent takes a negligible amount of time for computation and communication; (5) Each agent acts like a communication relay.

Assumptions (2) and (3), combined with the strategy presented in Section V, will guarantee that only one robot can occupy a location at a given time, for all times, and that no collisions among agents can appear. Finally, assumptions (4) and (5) insures that any two robots part of a connected component of the communication graph induced by relation $\sim$ can actually communicate.

Remark 1: The finite and purely discrete framework introduced above represents a formal abstraction of a realistic multi-robot scenario. Indeed, the set P can label the regions of a partitioned environment (obtained using, for example, triangulations or rectangular grids). The communication relation (2) can be the set of edges of the partition quotient graph, if two robots are restricted to communicate only when they are in adjacent regions. Alternatively, $(P, \sim)$ can be a visibility graph, if the inter-robot communication is based on line of sight. The "motion capacity" relation (1) captures our capability to design controllers guaranteeing that a robot can be either kept in a region for all times or driven to a neighbor region in finite time, regardless of the initial position of the robot in the current region. Computational efficient algorithms for the construction of such controllers were presented for triangular and rectangular partitions for robot dynamics ranging from fully actuated point-like robots to unicycles of non-negligible size [7], [9].

We consider the following problem:

Problem 1: Given a team of n agents with motion capabilities (1) and communication constraints (2), their initial non-overlapping positions in P, and a task specified as an "arbitrary" temporal and logic statement visiting locations from P, find individual control strategies for each agent that guarantee the accomplishment of the task.

As it will become clear in the next section, "arbitrary" temporal and logic statements will be formally defined as Linear Temporal Logic formulas. From our experience, this logic is expressive enough to capture most useful motion tasks, such as sequencing, convergence, surveillance, etc.

Problem 1 will be solved by designing one generic algorithm to be individually run by every agent, and agent-specific inputs for the generic algorithm on each agent. We make the natural assumption that first the memory of each agent is written, and then they are deployed in the environment and simultaneously powered on. Once the agents are programmed and started, they will autonomously evolve and can interact between themselves based on the imposed communication constraint. There is no central supervising controller for the team during the execution of the task, and this leads to a decentralized solution for the proposed problem.

Case study: For illustration, throughout this paper, we consider the example shown in Figure 1. The environment is assumed to be planar and partitioned in a set of rectangles, labelled from the set $P = \{p_1, \ldots, p_{16}\}$. We consider a team of three identical robots, with initial positions in regions p_1, p_3 and p_{13}, respectively. We assume that two robots can communicate when they are in rectangles that share an edge (rectangles diagonally placed do not communicate). We want to accomplish the following task:

"Do not visit any yellow region (p_8, p_{10}, p_{16}) until all these three regions are simultaneously visited, and always avoid the grey regions (p_4, p_6, p_{15})"

As we will see, such a task, although close to natural language, can be easily translated into an LTL formula. Note that the nature of the task clearly implies a cooperation in form of synchronization between robots (all yellow regions must be simultaneously visited).

Since the three robots are identical, we have $\alpha_1 = \alpha_2 = \alpha_3$ (we will simply denote these relations by α in further references to this case study). For simplicity, we will assume that each robot can stay inside a rectangle for all times, and move to any rectangle sharing an edge with the current rectangle. In other words, α is the adjacency relation between rectangles, and coincides with the communication relation $\sim$. Again, as mentioned in Remark 1, this "purely discrete" scenario is not necessarily restrictive. The robots can be, for example, fully actuated point like robots with polyhedral control constraints moving in the rectangulated environment. The "discrete" communication constraint can, for example, be seen as arising from the restriction that two robots can directly communicate only when the distance between them is at most 2.5.

III. Preliminaries

Definition 1: A transition system is a tuple $T = (Q, Q_0, \rightarrow, \Pi, \vDash)$, where Q is a set of states, $Q_0 \subseteq Q$ is a set of initial states, $\rightarrow \subseteq Q \times Q$ is a transition relation, Π is a finite set of atomic propositions (or observations), and $\vDash \subseteq Q \times \Pi$ is a satisfaction relation.

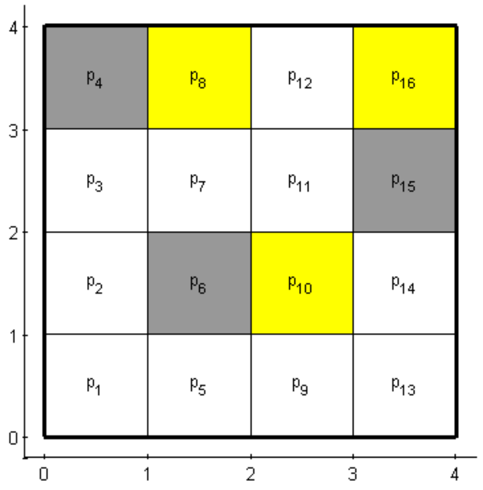

Fig. 1. Partition and task for the proposed example. The yellow regions should be simultaneously entered, and the grey regions should be always avoided.

For an arbitrary state $q \in Q$, let $\Pi_q = \{\pi \in \Pi \,|\, q \vDash \pi\}$, $\Pi_q \in 2^{\Pi}$, denote the set of all atomic propositions satisfied at q. A *trajectory* or *run* of T starting from q is an infinite sequence $r = r(1)r(2)r(3)\ldots$ with the property that $r(1) = q$, $r(i) \in Q$, and $(r(i), r(i+1)) \in \rightarrow$, for all $i \geq 1$. A trajectory $r = r(1)r(2)r(3)\ldots$ defines a *word* over set 2^{Π}, $w = w(1)w(2)w(3)\ldots$, where $w(i) = \Pi_{r(i)}$. The set of all words that can be generated by T is called the (ω-) language of T.

As already mentioned, we consider motion specifications given as formulas of Linear Temporal Logic (LTL) [4] [1]. A formal definition for the syntax and semantics of LTL formulas is beyond the scope of this paper. The LTL formulas are recursively defined over a set of atomic propositions Π, by using the standard boolean operators and a set of temporal operators. The boolean operators are $\neg$ (negations), $\vee$ (disjunction), $\wedge$ (conjunction), and some temporal operators that we use are: $\mathscr{U}$ (standing for "until"), $\Box$ ("always"), $\Diamond$ ("eventually"). LTL formulas are interpreted over infinite words over set 2^{Π}, as are those generated by transition system T. Assume that ϕ_1 and ϕ_2 are two LTL formulas over Π and w is a word produced by T. Formula $\phi_1 \mathscr{U} \phi_2$ intuitively means that (over word w) ϕ_2 will eventually become true and ϕ_1 is true until this happens. Formula $\Diamond\phi$ means that ϕ becomes eventually true, whereas $\Box\phi$ indicates that ϕ is true at all positions of w. More expressiveness can be achieved by combining the mentioned operators.

The mentioned syntax makes LTL very appealing for specifying motion tasks, and some supporting examples include reachability and obstacle avoidance tasks ("reach A eventually, while always avoiding B", written as $\Diamond A \wedge \Box \neg B$), convergence tasks ("reach A eventually and stay there for all future times" - $\Diamond\Box A$), etc. Moreover, if more robots are available, the attainment of disjoint regions at the same time might be of interest, as in "reach A and B eventually" ($\Diamond(A \wedge B)$).

[1]For technical reasons related to the abstraction from continuous to discrete robot dynamics [14], we restrict our attention to LTL_{-X}, a subclass of LTL that lacks the "next" temporal operator.

The specification from the Example introduced in Section II can be translated into the following LTL formula:

$$\phi = \neg(p_8 \vee p_{10} \vee p_{16}) \mathscr{U} (p_8 \wedge p_{10} \wedge p_{16}) \wedge \Box\neg(p_4 \vee p_6 \vee p_{15})$$

IV. Robot Models and Problem Reformulation

Using the definitions from Section III, we are now ready to reformulate Problem 1 formally.

The motion capabilities of each robot i, $i = 1, \ldots, n$, will be embedded in a transition system T_i with k states, each state corresponding to a location of set P. The transitions of every T_i will be trivially inherited from relation α_i, and the initial state will represent the initial deployment of agent i, $i = 1, \ldots, n$.

Each transition system T_i, $i = 1, \ldots, n$ has the form:

$$T_i = (Q_i, q_{0_i}, \rightarrow_i, \Pi_i, \vDash_i),\ i = 1, \ldots, n, \tag{3}$$

where $Q_i = P = \{p_1, \ldots, p_k\}$, $q_{0_i} \in Q$ is the initial location of agent i (a singleton), $\rightarrow_i = \alpha_i (\subseteq Q_i \times Q_i)$ is the transition relation, $\Pi_i = P$ is the set of locations (propositions for the LTL formula), and $\vDash_i$ is the trivial satisfaction relation $(q, \pi) \in \vDash_i$ if and only if $q = \pi$.

Note that the only differences between the transition systems T_i are given by their initial states and possibly by their transition relations (for agents with different movement capabilities). The motion of each agent is in fact abstracted to the sequence of locations it reaches, *i.e.,* to a sequence of symbols from P. Thus, a motion of agent i is a word produced by a run of T_i from equation (3), and the motion of the whole team of agents can be viewed as generating a word w over set 2^P (tuples of regions currently occupied by the n robots). Since the task is an LTL formula over set P, we can speak of satisfaction of this LTL formula by the team of agents.

Problem 1 can be now reformulated as:

Problem 2: Given a set of n transition systems T_i as in (3), a specification ϕ given as an LTL formula over their set of observations P, and a relation $\sim \in P \times P$ capturing the communication constraints among agents, find individual control strategies for each agent that guarantee the accomplishment of the task.

A control strategy will produce sequences of locations to be visited by an agent (a run of T_i). However, along such a run there will be some moments of communication with other agents, necessary for accomplishing a synchronized motion, as detailed in Section V.

Case study revisited: Returning to the proposed example, the motion of each agent is captured by a transition system with 16 states, each state corresponding to a rectangle from partition. The transitions are obviously given by adjacency relations between rectangles, plus self-transitions in every state, as resulted from relation α. The only difference between T_1, T_2 and T_3 is given by their initial states: $q_{0_1} = p_1$, $q_{0_2} = p_3$, and $q_{0_3} = p_{13}$.

V. Finding a Decentralizable Solution

In this section we will find a solution for the whole team of agents, in the form of a run of a transition system showing the succession of n-tuples of states to be visited. Although a similar idea was used in [16], we are now searching for a solution that can be distributed among individual agents, in contrast with a fully centralized one. We build this solution as follows: first, we construct a global transition system T_G capturing the behavior of the whole team. Second, we find a decentralizable run of T_G satisfying the LTL formula.

To formally define T_G, we first have to define two more functions, by using the set Q_G from Definition 2 (the set of all possible ordered n-tuples with elements from P):

$$\mathscr{M} : Q_G \times Q_G \rightarrow 2^{\{1,\dots,n\}}, \tag{4}$$

with $\mathscr{M}(q,q')$ giving the set of indices of agents that occupy different regions in configurations q and q', respectively (*e.g.*, if we have 3 agents, $\mathscr{M}((p_2,p_6,p_7),(p_2,p_7,p_5)) = \{2,3\}$).

Furthermore, let:

$$\mathscr{C} : Q_G \times \{1,\dots,n\} \rightarrow 2^{\{1,\dots,n\}}, \tag{5}$$

where $\mathscr{C}(q,i)$ is the set of agents communicating with robot i when configuration of the team is q. The communication mentioned here is not just the direct communication, but includes communication through all possible agents acting like relays. Function $\mathscr{C}$ is computed by using the relation $\sim$ and standard procedures from graph theory.

Definition 2: The transition system $T_G = (Q_G, Q_{G0}, \rightarrow_G, \Pi_G, \vDash_G)$ capturing the behavior of the group of n agents is defined as:

- $Q_G \subset Q_1 \times \ldots \times Q_n$ is defined by $(q_1,\dots,q_n) \in Q_G$ if and only if $q_i \neq q_j$ for $i \neq j$,
- $Q_{G0} = (q_{0_1},\dots,q_{0_n})$,
- $\rightarrow_G \subset Q_G \times Q_G$ is defined by $(q,q') \in \rightarrow_G$, with $q = (q_1,\dots,q_n)$ and $q' = (q'_1,\dots,q'_n)$, if and only if (1) $(q_i,q'_i) \in \rightarrow_i$, $i = 1,\dots,n$, (2) $\forall i,j = 1,\dots,n$ with $i \neq j$, if $q'_i = q_j$, then $q'_j \neq q_i$, and (3) $\mathscr{M}(q,q') \subseteq \mathscr{C}(q,i)$, $\forall i \in \mathscr{M}(q,q')$,
- $\Pi_G = P$,
- $\vDash_G \subset Q_G \times \Pi_G$ is defined by $((q_1,\dots,q_n),\pi) \in \vDash_G$ if $\pi \in \{q_1,\dots,q_n\}$.

In other words, the states of the transition system T_G capture all possible ways in which the k regions labelled with symbols from P are occupied by the n robots (cardinality of Q_G equals arrangements of k things taken n at a time). Configurations in which two agents overlap (occupy the same region) are excluded, thus guaranteeing the avoidance of inter-agent collisions. The possible motions of the team are modelled by the transition relation $\rightarrow_G$. A transition of T_G occurs when some agents synchronously[2] take allowed transitions (requirement (1) of Definition 2), and we exclude the case when two agents switch positions, since this could cause collision (requirement (2)). Moreover, requirement (3) shows that all moving agents should communicate among themselves. This last requirement is implied by the fact that only communicating agents can synchronize when changing their currently occupied locations. Note that the moving agents should communicate only while the team is in the configuration to be left (denoted by $q = (q_1,\dots,q_n)$ in Definition 2), and they are not restricted to communicate when the next configuration (denoted by $q' = (q'_1,\dots,q'_n)$) is reached. This is because the synchronization when leaving configuration q guarantees that the corresponding locations from q' are synchronously hit. In requirement (3), $i \in \mathscr{M}(q,q')$ can be arbitrarily chosen, since $\mathscr{C}(q,i_1) = \mathscr{C}(q,i_2)$, $\forall i_1,i_2 \in \mathscr{M}(q,q')$, and we call the set $\mathscr{C}(q,i)$ the *active communicating set*. Finally, each team configuration is equipped with n predicates enumerating the locations occupied by the agents (satisfied propositions), without explicitly specifying their exact position.

We now want to find a run of T_G whose produced word (over set 2^P) satisfies the LTL specification ϕ. Indeed, if we had such a run r, we would project it to the n agents (by splitting each of its n-tuples in its components) and we would obtain a run for each transition system T_i. Transition relation from T_G guarantees that for every single transition from these runs, the moving agents can communicate, and thus synchronize among themselves. However, there is no guarantee that the agents know when to take their next transition, simply because they might not know when the previously moving group completed their transition. For overcoming this, we introduce a requirement we call *transitivity of communication*, stating that a run of T_G can be decentralized if any two successive active communicating sets have nonempty intersection. Formally, a run $r = r(1)r(2)r(3)\ldots$ of T_G satisfies the transitivity of communication property if, for any position $i = 1,2,\dots$, we have:

$$\begin{aligned}&\mathscr{C}(r(i),j) \cap \mathscr{C}(r(i+1),k) \neq \emptyset,\\ &\forall j \in \mathscr{M}(r(i),r(i+1)),\ \forall k \in \mathscr{M}(r(i+1),r(i+2))\end{aligned} \tag{6}$$

In [14] we developed a tool inspired by model checking, which receives as inputs a transition system and an LTL formula over its set of propositions, and finds (when possible) a run of the transition system satisfying the LTL formula. The run has a special structure, namely a finite sequence of states called prefix, followed by an infinitely repeating finite sequence of states, called suffix. Moreover, there are no finite successive repetitions of a state in an obtained run. For details of this algorithm, we refer the interested reader to [14]. However, neither the tool from [14], nor a model checking algorithm can be used (or easily modified) for obtaining a run satisfying property (6). Therefore, we will cut transitions in T_G such that the newly obtained reduced transition system, called T_r, generates only runs satisfying (6).

Due to space constraints and the involved formalism, we only give the main idea for reducing T_G to T_r. We first find all the disjoint sets of communicating agents in initial state Q_{G0}. For each of these sets, a different T_r can be created as follows. Consider one of this sets to be the active

[2]Synchronization of transition systems is usually achieved through shared actions, or inputs [17]. However, including such synchronization actions in Definition 1 would unnecessarily complicate the notation.

communicating set, denoted by $\mathscr{C}_0$, and start with $T_r = T_G$. From all outgoing transitions of Q_{G0} in T_r, keep only the self-loop and those for which the set of moving agents is included in $\mathscr{C}_0$. Then, for all the new states that can be visited by taking these transitions, keep only the self-loops and the outgoing transitions for which the intersection between their active communicating set and $\mathscr{C}_0$ is nonempty. The process is repeated for each newly visited state, by updating $\mathscr{C}_0$ to be the active communicating set for the currently considered outgoing transition. The algorithm finishes when no new (not yet visited) states are reached.

We feed T_r and ϕ to the algorithm from [14], and if we obtain a run r, this is also guaranteed to be a run of T_G (because the set of transitions of T_r is a subset of transitions of T_G, fact which implies the inclusion of the language of T_r in the language of T_G). To choose among several possible runs, we select the one corresponding to a minimum number of robot movements (transitions in T_i, without counting self-loops) in prefix and first repeating sequence from suffix. However, such a run is not necessary optimal under the same criterion in T_G. If a run is not obtained, we construct another T_r, by considering another initial $\mathscr{C}_0$, and we search for a run in this T_r. If we do not obtain a run for at least one T_r we can build, we conclude that Problem 1 is infeasible. Assuming a run is obtained, we further have to actually decentralize it, as described in Section VI.

Case study revisited: The obtained T_G has 3360 states, and its initial state is $Q_{G0} = (p_1, p_3, p_{13})$. There are three communicating sets in Q_{G0}, namely 1,2,3 (*e.g.*, Q_{G0} can transit in (p_1, p_4, p_{13}), but cannot transit in (p_2, p_4, p_{13})). T_G has a total number of 55344 transitions, and its construction took about 45 seconds on a medium performance laptop. The first reduced transition system T_r is constructed by considering $\mathscr{C}_0 = \{3\}$, the obtained reduced transition system T_r has only 28583 transitions, and it was created in 64 seconds. A run satisfying the formula was obtained (in 23 seconds) and thus we don't have to create any other T_r. The obtained run is depicted in Figure 2. It has a prefix of length 7, namely the sequence of states (p_1, p_3, p_{13}) (p_1, p_3, p_9) (p_1, p_3, p_5) (p_2, p_3, p_1) (p_3, p_7, p_2) (p_7, p_{11}, p_3) (p_{11}, p_{12}, p_7), and a suffix consisting in infinitely many repetitions of configuration (p_{10}, p_{16}, p_8). By design, this run of T_G satisfies both the desired specification and the transitivity of communication property. Initially agent 3 moves towards agent 1, then they start to communicate and both move towards robot 2, and eventually all three agents reach configuration (p_{11}, p_{12}, p_7), from where they can enter the yellow regions at the same time.

VI. Decentralized Implementation

Assuming that a run r of T_G is obtained as described in Section V, we want to decentralize it among robots. We first design a generic algorithm to be run on each agent, and then, using run r, we produce agent-specific inputs for these algorithms.

Algorithm 1 presents the steps that are infinitely iterated by each robot. The main idea in decentralizing a run r is to have a *token* possessed by a single agent at any given time. The agent having the token initiates the next synchronous movement of some agents (next transition in the run r). Thus, there will be no simultaneous movements of non-communicating agents. Every agent keeps broadcasting (in its communication range) its identification (index) and the location it currently occupies, and listens for possible commands. There are two types of commands that an agent can receive: a synchronized moving command, showing the next location to be reached and the group of robots to synchronize with, and a token command, showing that the token is passed to the agent in question. Unless a moving command is received, each agent is stopped in the current location.

The memory of each agent is organized as a first-in first-out queue. An entry in this queue stores the following information: *position*, taking values in set $\{prefix, suffix\}$, and showing if the current configuration of the team is in the prefix or suffix; $\mathscr{M} \subseteq \{1, \ldots, n\}$, giving the set (indices) of moving agents for the next transition from r to be taken; $\mathscr{N}_p$ (an ordered tuple with $|\mathscr{M}|$ elements from P), containing the next location that each agent from $\mathscr{M}$ should visit; $next_{tok} \in \{1, \ldots, n\}$, giving the index of the agent that will receive the token once the currently scheduled transition from r is taken. The agent initiating the current transition detects its completion based on the following: either it receives the new locations occupied by agents in $\mathscr{M}$, or, if it cannot communicate anymore with some agents from $\mathscr{M}$, it concludes that those agents completed their moving command.

Algorithm 1 Operations iterated by each agent $i = 1, \ldots, n$

```
► listen and execute received commands
► broadcast index $i$ and location occupied
if $token = 1$ then
    ► read first entry from queue memory: $position$, $\mathscr{M}$, $\mathscr{N}_p$, $next_{tok}$
    ► send to agents from $\mathscr{M}$ the next location to be visited (corresponding position from $\mathscr{N}_p$), and set to synchronize with ($\mathscr{M}$)
    ► wait for agents from $\mathscr{M}$ to complete their task
    if $position = prefix$ then
        ► delete first entry from queue memory
    else
        ► move first entry from queue memory to end
    end if
    ► send $token$ to agent $next_{tok}$
end if
```

The robot-specific inputs for Algorithm 1 are found by scanning the successive team configurations and active communicating sets encountered along the run r. Due to space constraints, we omit the details, and we refer the interested reader to the technical report from http://iasi.bu.edu/∼software/LTL-decentr.htm.

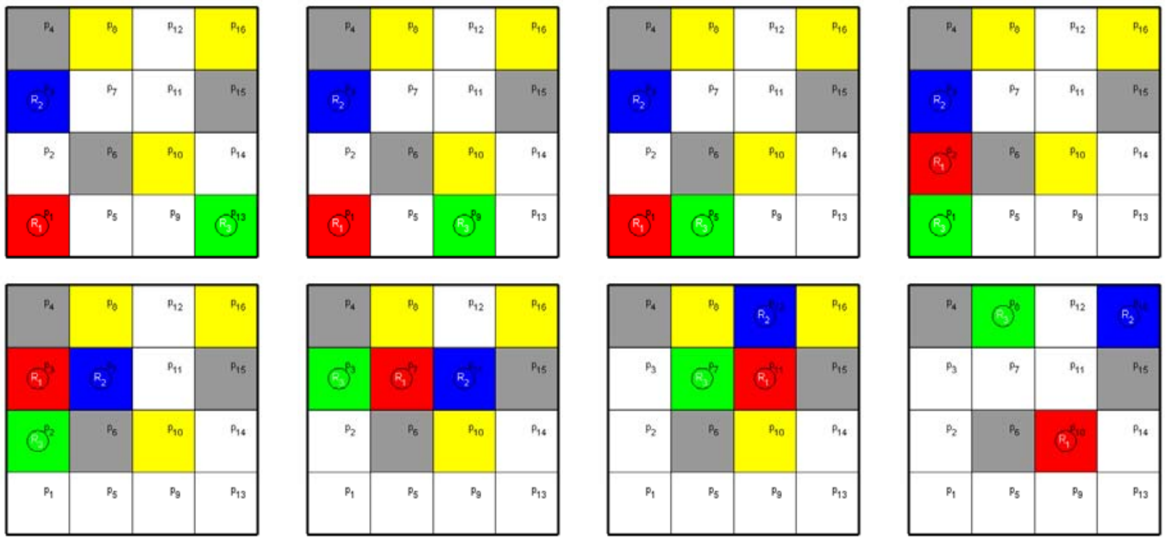

Fig. 2. Successive configurations of the team for the proposed example. The regions occupied by the three robots are red, blue and green, respectively. The regions to be simultaneously reached are yellow, and the regions to be always avoided are grey.

VII. Conservatism and Complexity

Our approach to solving Problem 1 is conservative because of three main reasons. First, the assumptions from Section II introduce conservativeness with respect to movement capabilities and communication constraints. Second, we assume that only communicating agents can move at the same time, rather than allowing independent movements of agents from different communication sets. Third, reducing T_G to T_r as in Section V can imply that no solution satisfying property (6) is found, although such a solution might have existed in T_G.

From the complexity point of view, the bottleneck of the presented approach is the part from Section V, because T_G has $k!/(k-n)!$ states. Not only creating and reducing T_G to T_r raises complexity problems, but also finding a run in T_r. In order to find a run, a Büchi automaton (accepting infinite words satisfying the LTL formula) is involved, and its size is exponentially upper bounded in the size of the LTL formula.

VIII. Conclusions

We presented a fully automated computational framework for decentralized deployment of a team of robots from task specifications given in high-level and rich language consisting of temporal logic statements about visiting regions of interest in a partitioned environment. While we focused on a purely discrete problem, the implementation of the algorithm is immediate for robots with continuous dynamics moving in environments with triangular or rectangular partitions, and with communication constraints induced by physical proximity. The approach was implemented as a software package under Matlab, which is freely downloadable from http://iasi.bu.edu/∼software/LTL-decentr.htm.

References

[1] J. C. Latombe, *Robot Motion Planning*. Kluger Academic Pub., 1991.

[2] V. Gazi and K. M. Passino, "Stability analysis of swarms," *IEEE Transactions on Automatic Control*, vol. 48, no. 4, pp. 692–696, 2003.

[3] N. E. Leonard and E. Fiorelli, "Virtual leaders, artificial potentials, and coordinated control of groups," in *40th IEEE Conference on Decision and Control*, Orlando, FL, December 2001, pp. 2968 – 2973.

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

[5] Z. Manna and A. Pnueli, *The temporal Logic of Reactive and Concurrent Systems: Specification*. Berlin: Springer-Verlag, 1992.

[6] L. Habets and J. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope," *Automatica*, vol. 40, pp. 21–35, 2004.

[7] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot planning and control in polygonal environments," *IEEE Trans. on Robotics*, vol. 21, no. 5, pp. 864–874, 2005.

[8] C. Belta and L. Habets, "Control of a class of nonlinear systems on rectangles," *IEEE Transactions on Automatic Control*, vol. 51, no. 11, pp. 1749 – 1759, 2006.

[9] M. Kloetzer and C. Belta, "A framework for automatic deployment of robots in 2d and 3d environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[10] M. Antoniotti and B. Mishra, "Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers," in *IEEE International Conference on Robotics and Automation*, May 1995.

[11] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multiagent motion tasks based on LTL specifications," in *43rd IEEE Conference on Decision and Control*, December 2004.

[12] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-robot motion planning: A timed automata approach," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004, pp. 4417–4422.

[13] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: a temporal logic approach," in *Proceedings of the 2005 IEEE Conference on Decision and Control*, Seville, Spain, December 2005.

[14] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from LTL specifications," in *Hybrid Systems: Computation and Control: 9th International Workshop*, ser. Lecture Notes in Computer Science, J. Hespanha and A. Tiwari, Eds. Springer Berlin / Heidelberg, 2006, vol. 3927, pp. 333 – 347.

[15] M. Mukund, "From global specifications to distributed implementations," in *Synthesis and control of discrete event systems*. Kluwer, 2002, pp. 19–34.

[16] M. Kloetzer and C. Belta, "LTL planning for groups of robots," in *IEEE International Conference on Networking, Sensing, and Control*, Ft. Lauderdale, FL, 2006.

[17] R. Milner, *Communication and concurrency*. Englewood CliDs, NJ: Prentice-Hall, 1989.