Automatica 70 (2016) 239–248

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

Multi-agent planning under local LTL specifications and event-based synchronization☆

Jana Tumova, Dimos V. Dimarogonas

School of Electrical Engineering, KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden

ARTICLE INFO

Article history:
Received 3 March 2015
Received in revised form
3 March 2016
Accepted 18 March 2016
Available online 23 April 2016

Keywords:
Temporal logic
Finite state machines
Formal verification
Path planning
Synchronization
Decentralized control
Robot control

ABSTRACT

We study the problem of plan synthesis for multi-agent systems, to achieve complex, high-level, long-term goals that are assigned to each agent individually. As the agents might not be capable of satisfying their respective goals by themselves, requests for other agents' collaborations are a part of the task descriptions. We consider that each agent is modeled as a discrete state-transition system and its task specification takes a form of a linear temporal logic formula. A traditional automata-based approach to multi-agent plan synthesis from such specifications builds on centralized team planning and full team synchronization after each agents' discrete step, and thus suffers from extreme computational demands. We aim at reducing the computational complexity by decomposing the plan synthesis problem into finite horizon planning problems that are solved iteratively, upon the run of the agents. We introduce an event-based synchronization that allows our approach to efficiently adapt to different time durations of different agents' discrete steps. We discuss the correctness of the solution and find assumptions, under which the proposed iterative algorithm leads to provable eventual satisfaction of the desired specifications.

1. Introduction

In recent years, a considerable amount of attention has been devoted to synthesis of robot controllers for complex, high-level missions, such as "periodically survey regions *A*, *B*, *C*, in this order, while avoiding region *D*", specified as temporal logic formulas. Many of the suggested solutions to variants of this problem rely on a hierarchical procedure (Bhatia, Maly, Kavraki, & Vardi, 2011; Kloetzer & Belta, 2008; Kress-Gazit, Fainekos, & Pappas, 2009; Wongpiromsarn, Topcu, & Murray, 2010): First, the dynamics of the robotic system is abstracted into a finite transition system using e.g., sampling or cell decomposition methods. Second, leveraging ideas from formal verification, a discrete plan that meets the mission is synthesized. Third, the plan is translated into a controller for the original system. In this work, we focus on a multi-agent version of the above problem. We consider a heterogeneous team of robots, that are assigned a temporal logic mission each. As the robots may not be able to accomplish their missions without the help of the others, the specifications may contain requirements on the other team members' behavior. For instance, consider a warehouse solution with two mobile robots. A part of the first robot's mission is to load an object in region *A*, but it is not able to load it by itself. Therefore, the mission also includes a task for the second robot, to help loading. The goal of this paper is to efficiently synthesize a plan for each agent, such that each agent's mission is met. We follow the hierarchical approach to robot controller synthesis as outlined above and we narrow our attention to the second step of the approach, i.e., to generating discrete plans. The application of the algorithm that we propose is, however, not restricted to discrete systems: For the first step of the hierarchical approach, numerous methods for discrete modeling of robotic systems can be used (see, e.g., Kloetzer & Belta, 2008; Kress-Gazit et al., 2009; LaValle, 2006; Wongpiromsarn et al., 2010 and the references therein); for the third step, low-level controllers exist that can drive a robot from any position within a region to a goal region (see, e.g., Belta & Habets, 2006). The agents can, but do not have to, mutually synchronize after the execution of their respective discrete steps. The desired plans thus comprise not only of the agents' discrete steps to be taken, but also their synchronizations. Besides the satisfaction of all agents' missions, our goal is to avoid unnecessary synchronization in order to improve the team performance.

☆ This work was supported by the EU STREP RECONFIG, and by the H2020 ERC Starting Grant BUCOPHSYS. The material in this paper was partially presented at the 2014 American Control Conference, June 4–6, 2014, Portland, OR, USA. This paper was recommended for publication in revised form by Associate Editor Jan Komenda under the direction of Editor Christos G. Cassandras.

E-mail addresses: tumova@kth.se (J. Tumova), dimos@kth.se (D.V. Dimarogonas).

http://dx.doi.org/10.1016/j.automatica.2016.04.006
0005-1098/

As a mission specification language, we use Linear Temporal Logic (LTL), for its resemblance to natural language (Jing, Finucane, Raman, & Kress-Gazit, 2012), and expressive power. Here, we built LTL formulas over services, i.e., events of interest associated with execution of certain actions rather than over atomic propositions, i.e., inherent properties of the system states. Instead of evaluation of the specification as a conjunction of LTL formulas over the whole team behaviors, we propose the notion of satisfaction of an LTL formula from local perspective. This way, the problem of finding a collective team behavior is decomposed into several subproblems, enabling us to avoid the straightforward, but expensive fully centralized planning. The contribution of this paper can be summarized as the introduction of an efficient, iterative, finite horizon planning technique in the context of bottom-up plan synthesis for multi-agent systems from local LTL specifications. To our best knowledge, such an approach has not been taken to address the multi-agent LTL planning before. Our algorithm is adaptive in the sense that even if the real behavior of the team is not as planned due to unpredictable time durations of the agents' steps, the event-based synchronization and replanning still guarantees the satisfaction of all the tasks. This feature can be especially beneficial in heterogeneous multi-robot motion and task planning problems, where individual robots traverse their common environment at different speeds. This paper builds on our earlier work in Tumova and Dimarogonas (2014). In addition, it relaxes the assumption that the agents synchronize after every discrete step of theirs and introduces the event-based synchronization and replanning.

Multi-agent planning from temporal logic specification has been explored in several recent works. Planning from computational tree logic was considered in Quottrup, Bak, and Zamanabadi (2004), whereas in Kloetzer, Ding, and Belta (2011) and Loizou and Kyriakopoulos (2005), the authors focus on planning behavior of a team of robots from a single, global LTL specification. A fragment of LTL has been considered as a specification language for vehicle routing problems in Karaman and Frazzoli (2011), and a general reactivity LTL fragment has been used in Wiltsche, Ramponi, and Lygeros (2013). Decentralized control of a robotic team from local LTL specification with communication constraints is proposed in Filippidis, Dimarogonas, and Kyriakopoulos (2012). However, the specifications there are truly local and the agents do not impose any requirements on the other agents' behavior. Thus, the focus of the paper is significantly different to ours. As opposed to our approach, in Chen, Ding, Stefanescu, and Belta (2012) and Ulusoy, Smith, Ding, Belta, and Rus (2013), a top-down approach to LTL planning is considered; the team is given a global specification and an effort is made to decompose the formula into independent local specifications that can be treated separately for each robot. In Guo and Dimarogonas (2015), bottom-up planning from LTL specifications is considered, and a partially decentralized solution is proposed that takes into account only clusters of dependent agents instead of the whole group. A huge challenge of the previous approach is its extreme computational complexity, which we tackle in this work by applying receding horizon approach to multi-agent planning. Receding horizon approach was leveraged also in Wongpiromsarn et al. (2010) to cope with uncertain elements in an environment in single-robot motion planning. To guarantee the satisfaction of the formula, we use an attraction-type function that guides the individual agents towards a progress within a finite planning horizon; similar ideas were used in Ding, Belta, and Cassandras (2010) and Svorenova, Tumova, Barnat, and Cerna (2012) for a single-agent LTL planning to achieve a locally optimal behavior.

The rest of the paper is structured as follows. In Section 2, we fix the preliminaries. Section 3 introduces the problem and summarizes our approach. In Section 4, the details of the solution are provided. In Section 5, we provide analysis and discussion of the solution. We present simulation results in Section 6, and we conclude in Section 7.

2. Preliminaries

Given a set S, let 2^{S}, and S^{ω} denote the set of all subsets of S, and the set of all infinite sequences of elements of S, respectively. A finite or infinite sequence of elements of S is called a finite or infinite word over S, respectively. The ith element of a word w is denoted by $w(i)$. A subsequence of an infinite word $w = w(1)w(2)\ldots$ is a finite or infinite sequence of its elements $w(i_1)w(i_2)\ldots$, where $\forall 1 \leq j.\ 1 \leq i_j \leq i_{j+1}$. A factor of w is a continuous, finite or infinite, subsequence $w(i)w(i+1)\ldots$, where $1 \leq i$. A prefix of w is a finite factor starting at $w(1)$, and a suffix of w is an infinite factor. $\mathbb{N}$ and $\mathbb{R}_0^+$ denote positive integers and non-negative real numbers, respectively.

A *transition system (TS)* is a tuple $\mathcal{T} = (S, s_{init}, A, T)$, where S is a finite set of states; $s_{init} \in S$ is the initial state; A is a finite set of actions; and $T \subseteq S \times A \to S$ is a partial deterministic transition function. For simplicity, we denote a transition $T(s, \alpha) = s'$ by $s \xrightarrow{\alpha} s'$. A *trace* of $\mathcal{T}$ is an infinite alternating sequence of states and actions $\tau = s_1\alpha_1 s_2\alpha_2\ldots$, such that $s_1 = s_{init}$, and for all $i \geq 1$, $s_i \xrightarrow{\alpha_i} s_{i+1}$. A *trace fragment* $\hat{\tau}$ is a finite factor of a trace τ that begins and ends with a state.

A *linear temporal logic (LTL) formula* ϕ over the set of atomic propositions Π is defined inductively: (i) $\pi \in \Pi$ is a formula, and (ii) if ϕ_1 and ϕ_2 are formulas, then $\phi_1 \vee \phi_2$, $\neg\phi_1$, $\mathsf{X}\phi_1$, $\phi_1 \mathsf{U} \phi_2$, $\mathsf{F}\phi_1$, and $\mathsf{G}\phi_1$ are each a formula, where $\neg$ and $\vee$ are standard Boolean connectives, and X, U, F, and G are temporal operators. The semantics of LTL are defined over infinite words over 2^{Π}. $\pi \in \Pi$ is satisfied on $w = \varpi_1\varpi_2\ldots$ if $\pi \in \varpi_1$. $\mathsf{X}\phi$ holds true if ϕ is satisfied on the word that begins in the next position ϖ_2, $\phi_1 \mathsf{U} \phi_2$ states that ϕ_1 has to be true until ϕ_2 becomes true, and $\mathsf{F}\phi$ and $\mathsf{G}\phi$ are true if ϕ holds on w eventually, and always, respectively. We denote the satisfaction of ϕ on a word w as $w \models \phi$. The set of all words accepted by an LTL formula ϕ is $\mathcal{L}(\phi)$. For full details see, e.g., Baier and Katoen (2008).

An *automaton* is a tuple $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, F)$, where Q is a finite set of states; $q_{init} \in Q$ is the initial state; Σ is an input alphabet; $\delta \subseteq Q \times \Sigma \times Q$ is a non-deterministic transition relation; and F is an accepting condition. It is *deadlock-free* if $\forall q \in Q, \sigma \in \Sigma.\ \delta(q, \sigma) \neq \emptyset$. We define the set of states $\hat{\delta}^k(q)$ reachable from $q \in Q$ in exactly k steps inductively as $\hat{\delta}^0(q) = \{q\}$, and $\hat{\delta}^k(q) = \bigcup_{q' \in \hat{\delta}^{k-1}(q)} \{q'' \mid \exists \sigma \in \Sigma.\ (q', \sigma, q'') \in \delta\}, \forall k \geq 1$. A *Büchi automaton (BA)* is an automaton with the accepting condition $F \subseteq Q$. A *run* of the BA $\mathcal{B}$ from $q_1 \in Q$ *over* $w = \sigma_1\sigma_2\ldots \in \Sigma^{\omega}$ is a sequence $\rho = q_1 q_2\ldots$, such that $\forall i \geq 1.\ (q_i, \sigma_i, q_{i+1}) \in \delta$. A run ρ is *accepting* if it intersects F infinitely many times. A word w is *accepted* by $\mathcal{B}$ if there exists an accepting run over w from the state q_{init}. The set of all words accepted by $\mathcal{B}$ is $\mathcal{L}(\mathcal{B})$. Any automaton $(Q, q_{init}, \Sigma, \delta, F)$ can be viewed as a graph (V, E) with the vertexes $V = Q$ and the edges E given by δ in the expected way. The standard notation then applies: A *path* is a finite alternating sequence of states and transition labels $q_i\sigma_i q_{i+1}\ldots q_{k-1}\sigma_{k-1}q_k$, such that $\forall i \leq j < k.\ (q_j, \sigma_j, q_{j+1}) \in \delta$. $\mathrm{dist}(q, q')$ denotes the length of the *shortest path* between q and q', i.e., the minimal number of states in a path $q\ldots q'$. If no such path exists, then $\mathrm{dist}(q, q') = \infty$. If $q = q'$, then $\mathrm{dist}(q, q') = 0$. The shortest path can be computed using Dijkstra algorithm (see, e.g., Cormen, Stein, Rivest, & Leiserson, 2001).

3. Problem formulation and approach

Two general viewpoints can be taken in multi-agent planning: either the system acts as a team with a common goal, or the agents have their own, independent tasks. Although we permit each agent's task to involve requirements on the others, we adopt the second viewpoint; to decide whether the agents' tasks are met,

we do not look at the global team behavior. In contrast, we propose the concept of local specification satisfaction to determine whether an agent's task is fulfilled from its own perspective. Our goal is to synthesize a plan for each agent that comprises of (i) the actions to be executed and (ii) the synchronization requests imposed on the other agents. Together, the strategies have to ensure the local satisfaction of each of the task specifications regardless of the time duration of the planned action executions.

We consider N agents (e.g., robots in a partitioned environment). Each agent $i \in \mathcal{N}$, where $\mathcal{N} = \{1, \ldots, N\}$ has action execution and synchronization capabilities.

Action execution capabilities. The agent i's action execution capabilities are modeled as a TS $\mathcal{T}_i = (S_i, s_{init,i}, A_i, T_i)$, whose states correspond to states of the agents (e.g., the locations of the robots in the regions of their environment). The actions A_i represent abstractions of the agent's low-level controllers, and the transitions T_i correspond to the agent's capabilities to execute the actions (e.g., the ability of the robots to move between two regions). The traces are, roughly speaking, the abstractions of the agents' long-term behaviors (e.g., the robots' trajectories). For the simplicity of the presentation, we assume that each state $s \in S_i$ is reachable from all states $s' \in S_i$ via a sequence of transitions. Each of the agents' action executions takes a certain amount of time. Given a trace $\tau = s_1\alpha_1 s_2\alpha_2 \ldots$ of $\mathcal{T}_i$, we denote by $\Delta_{\alpha_j} \in \mathbb{R}_0^+$ the time *duration* of the transition $s_j \xrightarrow{\alpha_j} s_{j+1}$. Note that a transition duration is arbitrary and unknown prior its execution, and that the execution of the same action α may induce different transition durations in its different instances on a trace. The durations are not explicitly modeled in the TS $\mathcal{T}_i$ due to the fact that they are unknown, but, as we will discuss later on, their history plays an important role in defining the semantics of the agent's behaviors, their interactions and satisfaction of their tasks.

Synchronization capabilities. The agents have also the ability to *synchronize*, i.e., to wait for each other and to proceed with the further execution simultaneously. Through the simultaneous execution of certain transitions, the agents have the ability to collaborate (e.g., an agent loading heavy goods may need a second agent simultaneously helping to load it). The synchronization is modeled through the *synchronization requests*, but the particular implementation of the synchronization scheme is beyond the scope of this paper. While being in a state s, an agent i can send a request $sync_i$ to the set of agents $\mathcal{N}$ notifying them that it is ready to synchronize. Then, before proceeding with the execution of any action $\alpha \in A_i$, it has to wait till $sync_{i'}$ has been sent by each agent $i' \in \mathcal{N}$, i.e., till the moment when each agent $i' \in \mathcal{N}$ is ready to synchronize, too. The synchronization is immediate once each of the agents $i' \in \mathcal{N}$ has sent its request $sync_{i'}$ and all agents in $\mathcal{N}$ start executing the next action at the same time. Alternatively, an agent i indicates that it does not need to synchronize through requesting $nosync_i$. The set of all synchronization requests of an agent i is thus $Sync_i = \{sync_i, nosync_i\}$. For simplicity, we assume that each agent sends a synchronization request instantly once it completes an action execution and that it starts executing an action instantly once it synchronizes with the other agents. In other words, no time is spent idling in our system model. This does not prevent us to capture the agents' abilities of staying in their respective states. Instead of idling, we include a so-called self-loop $s \xrightarrow{\alpha} s$ for some $\alpha \in A_i$, all $s \in S_i$, and all $\mathcal{T}_i$, $i \in \mathcal{N}$. Given a trace $\tau_i = s_{i,1}\alpha_{i,1}s_{i,2}\alpha_{i,2} \ldots$ of $\mathcal{T}_i$, we denote by $\Delta_{s_{i,j}} \in \mathbb{R}_0^+$ the time duration of the synchronization requested in the state $s_{i,j}$. Note that if the request $nosync_i$ has been sent in $s_{i,j}$, then $\Delta_{s_{i,j}} = 0$.

Remark 1. In order to accommodate synchronization with a subset of agents, we can parametrize the synchronization requests $Sync_i = \{sync_i(M) \mid \{i\} \subseteq M \subseteq \mathcal{N}\}$. The use of such a definition is discussed later in Remark 3.

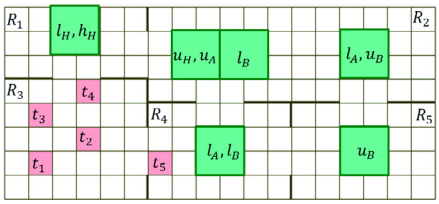

Fig. 1. An example of an environment partitioned into cells with rooms $R_1, \ldots, R_5$. Loading/unloading points are in green (light). Simple tasks $t_1, \ldots, t_5$ can be executed in purple (dark) regions. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Services. Each of the agents' tasks is given in terms of temporal requirements on events of interest, which we call *services*. The set of all services that can be *provided* by an agent $i \in \mathcal{N}$ is Π_i. Services are provided within the agents' transitions; each action $\alpha \in A_i$ is *labeled* either with (i) a service set $\varpi \in 2^{\Pi_i}$ provided upon the execution of α, or (ii) a special *silent* service set $\mathcal{E}_i = \{\varepsilon_i\}$, $\varepsilon_i \notin \Pi_i$ indicating that α is not associated with any event of interest. Hence, two additional components of the agent i's model are the set of all available services Π_i and the action-labeling function $L_i : A_i \to 2^{\Pi_i} \cup 2^{\mathcal{E}_i}$. Note that we specifically distinguish between a silent service set $\mathcal{E}_i$ and an empty service set $\{\}$. The self-loops of type $s \xrightarrow{\alpha} s$ introduced above to model staying are naturally labeled with $\mathcal{E}_i$. Without loss of generality, we assume that $\Pi_i \cap \Pi_{i'} = \emptyset$, for all $i, i' \in \mathcal{N}$, $i \neq i'$. As it will become clear later, the choice of non-silent and silent services in place of the traditional atomic propositions is motivated by the nature of multi-agent planning, where the agents are concerned about each other only at selected times, e.g., when collaboration is required.

Finally, we model an agent $i \in \mathcal{N}$ as the tuple $\mathcal{M}_i = (\mathcal{T}_i, Sync_i, \Pi_i, L_i)$.

Example 1. An illustrative example of three mobile robots in a common partitioned workspace is depicted in Fig. 1. The agents can transit between the adjacent cells that are not separated by a wall or stay where they are. The former transitions are labeled with $\mathcal{E}_i$, while non-silent services are associated with some of the latter ones. Namely, agent 1 can load (l_H, l_A, l_B), carry, and unload (u_H, u_A, u_B) a heavy object H or a light object A, B, in the green cells. Agent 2 can help to load object H (h_H), and execute simple tasks in the purple regions ($t_1 - t_5$). Agent 3 is capable of taking a snapshot of the rooms ($s_1 - s_5$) when being present in there.

Behaviors. The *behavior* of an agent i is defined by the actions it executes, its synchronizations with the other agents, and the time instants when the action executions and the synchronizations take place.

Definition 1 (*Behavior*)**.** A behavior of an agent i is a tuple $\beta_i = (\tau_i, \gamma_i, \mathbb{T}_i)$, where $\tau_i = s_{i,1}\alpha_{i,1}s_{i,2}\alpha_{i,2} \ldots$ is a trace of $\mathcal{T}_i$; $\gamma_i = r_{i,1}r_{i,2} \ldots$ is a *synchronization sequence*, where $r_{i,j} \in Sync_i$ is the synchronization request sent at the state $s_{i,j}$; and $\mathbb{T}_i = t_{s_{i,1}}t_{\alpha_{i,1}}t_{s_{i,2}}t_{\alpha_{i,2}} \ldots$ is a non-decreasing *behavior time sequence*, where $t_{s_{i,j}}$ is the time instant when the synchronization request $r_{i,j}$ was sent, and $t_{\alpha_{i,j}}$ is the time instant when the action $\alpha_{i,j}$ started being executed. The following hold: $t_{s_{i,1}} = 0$, and for all $j \geq 1$, $t_{s_{i,j+1}} - t_{\alpha_{i,j}} = \Delta_{\alpha_{i,j}}$, and $t_{\alpha_{i,j}} - t_{s_{i,j}} = \Delta_{s_{i,j}}$.

The notion of behavior, however, does not reflect the above described synchronization scheme. To that end, we define *compatible behaviors*. In what follows, the behavior of $i \in \mathcal{N}$ is denoted by $\beta_i = (\tau_i, \gamma_i, \mathbb{T}_i)$, where $\tau_i = s_{i,1}\alpha_{i,1}s_{i,2}\alpha_{i,2} \ldots$, $\gamma_i = r_{i,1}r_{i,2} \ldots$, and $\mathbb{T}_i = t_{s_{i,1}}t_{\alpha_{i,1}}t_{s_{i,2}}t_{\alpha_{i,2}} \ldots$.

Definition 2 (*Compatible Behaviors*)**.** A set of behaviors of the agents in $\mathcal{N}$ is *compatible*, if the following holds for all $i \in \mathcal{N}$, and $j \geq 1$: Suppose that $r_{i,j} = sync_i$. Then for all $i' \in \mathcal{N}$, there exists a *matching index* $j' \geq 1$, such that $r_{i',j'} = sync_{i'}$, and $t_{\alpha_{i,j}} = t_{\alpha_{i',j'}}$. Furthermore, there exists $i' \in \mathcal{N}$, such that $t_{s_{i',j'}} = t_{\alpha_{i',j'}}$, i.e., such that $\Delta_{s_{i',j'}} = 0$, for the matching index j'.

Suppose that the trace τ_i, the synchronization sequence γ_i, and the transition time durations $\Delta_{\alpha_{i,1}}, \Delta_{\alpha_{i,2}}, \ldots$ are all fixed for all $i \in \mathcal{N}$. Then there exists at most one collection of synchronization time durations $\Delta_{s_{i,1}}\Delta_{s_{i,2}} \ldots, i \in \mathcal{N}$ that yields a set of compatible behaviors $\{\beta_i = (\tau_i, \gamma_i, \mathbb{T}_i) \mid i \in \mathcal{N}\}$. In other words, if the agents follow the synchronization scheme, the existence and the values of their synchronization time durations and hence also their behavior time sequences are uniquely determined by τ_i, γ_i, and $\Delta_{\alpha_{i,1}}, \Delta_{\alpha_{i,2}}, \ldots$ given for all $i \in \mathcal{N}$.

Specifications. The individual agents' tasks may concern the respective agent's services as well as the services of the others. Formally, each of the agents is given an LTL formula ϕ_i over $\boldsymbol{\Pi}_i = \bigcup_{i' \in D_i} \Pi_{i'}$, for some $\{i\} \subseteq D_i \subseteq \mathcal{N}$. Loosely speaking, the satisfaction of an agent's formula depends on, and only on the behavior of the subset of agents D_i, including the agent itself.

Example 1 (*Continued*)**.** The robots are assigned the following collaborative tasks. Agent 1 needs the help of agent 2 with loading the heavy object. Then, it should carry the heavy object to an unloading point and unload it. Then, it should periodically load and unload both light objects ($\phi_1 = \mathsf{F}(l_H \wedge h_H \wedge \mathsf{X}\, u_H \wedge \bigwedge_{i \in \{A,B\}} \mathsf{GF}\,(l_i \wedge \mathsf{X} u_i))$). Agent 2 should periodically execute the simple tasks $t_1, \ldots, t_5$, in this order. Furthermore, it requests agent 3 to witness the execution t_5, by taking a snapshot of the room R_4 at the moment of the execution ($\phi_2 = \mathsf{GF}\,(t_1 \wedge \mathsf{X}\,(t_2 \wedge \mathsf{X}\,(t_3 \wedge \mathsf{X}\,(t_4 \wedge \mathsf{X}\, t_5 \wedge s_4)))))$. Agent 3 should patrol rooms R_2, R_4, R_5 ($\phi_3 = \bigwedge_{i \in \{2,4,5\}} \mathsf{GF}\, s_i$).

Let us now introduce the notation needed for formalizing the specification satisfaction. Consider for a moment a single agent $\mathcal{M}_i = (\mathcal{T}_i, Sync_i, \Pi_i, L_i)$, and its behavior β_i, where, for simplicity of notation in the following paragraphs, we use $\beta_i = (\tau, \gamma, \mathbb{T})$, where $\tau = s_1\alpha_1 s_2\alpha_2 \ldots$, and $\mathbb{T} = t_{s_1} t_{\alpha_1} t_{s_2} t_{\alpha_2} \ldots$. We denote by $v_\tau = \varpi_1\varpi_2 \ldots = L_i(\alpha_1)L_i(\alpha_2) \ldots \in (2^{\Pi_i} \cup 2^{\mathcal{E}_i})^\omega$ the unique *service set sequence* associated with τ. The *word* w_τ produced by τ is the subsequence of the non-silent elements of this sequence; $w_\tau = \varpi_{\iota_1}\varpi_{\iota_2} \ldots \in (2^{\Pi_i})^\omega$, such that $\varpi_1, \ldots, \varpi_{\iota_1 - 1} = \mathcal{E}_i$, and for all $j \geq 1$, $\varpi_{\iota_j} \neq \mathcal{E}_i$ and $\varpi_{\iota_j+1}, \ldots, \varpi_{\iota_{j+1}-1} = \mathcal{E}_i$. With a slight abuse of notation, we use $\mathbb{T}(\tau) = t_1 t_2 \ldots = t_{\alpha_1} t_{\alpha_2} \ldots$ to denote the *trace time sequence*, i.e., the subsequence of $\mathbb{T}$ when the (both silent and non-silent) services are provided. Furthermore, $\mathbb{T}(w_\tau) = t_{\iota_1} t_{\iota_2} \ldots$ denotes the *word time sequence*, i.e., the subsequence of $\mathbb{T}(\tau)$ that corresponds to the times when the non-silent services are provided. The word w_τ and the word time sequence $\mathbb{T}(w_\tau)$ might be finite as well as infinite. Since in this work we are interested in infinite, recurrent behaviors, we will consider as *valid* traces only those that produce infinite words. Consider a trace τ of $\mathcal{T}_i$ with the service set sequence $v_\tau = \varpi_1\varpi_2 \ldots$, and the trace time sequence $\mathbb{T}(\tau) = t_1 t_2 \ldots$. The service set $v_\tau(t) \in 2^{\Pi_i} \cup 2^{\mathcal{E}_i}$ provided at time $t \in \mathbb{R}_0^+$ is $v_\tau(t) = \varpi_j$ if $t = t_j$ for some $j \geq 1$, and $\mathcal{E}_i$ otherwise. The satisfaction of each LTL formula ϕ_i is interpreted locally, from the agent i's point of view, based on the word w_{τ_i} it produces and on the services of agents $i' \in D_i$ provided at the time instances $\mathbb{T}(w_{\tau_i})$ when i provides a non-silent service set itself.

Definition 3 (*Local LTL Satisfaction*)**.** Let τ be a trace of $\mathcal{T}_i$ with the word time sequence $\mathbb{T}(w_\tau) = t_{\iota_1} t_{\iota_2} \ldots$. The *word* produced by a set of compatible behaviors $\mathfrak{B}_i = \{\beta_{i'} \mid i' \in D_i\}$ is $w_{\mathfrak{B}_i} = \omega_{\iota_1}\omega_{\iota_2} \ldots$, where $\omega_{\iota_j} = \left(\bigcup_{i' \in D_i} v_{\tau_{i'}}(t_{\iota_j})\right) \cap \boldsymbol{\Pi}_i$, for all $j \geq 1$. The set of behaviors $\mathfrak{B}_i$ is *valid* if $w_{\mathfrak{B}_i}$ is infinite. The set of compatible behaviors $\mathfrak{B}_i$ *locally satisfies* ϕ_i for the agent i, $\mathfrak{B}_i \models \phi_i$, iff $\mathfrak{B}_i$ is valid and $w_{\mathfrak{B}_i} \models \phi_i$.

Note that even if $\mathfrak{B} = \mathfrak{B}_i = \mathfrak{B}_j$, it may be the case that $w_{\mathfrak{B}_i} \neq w_{\mathfrak{B}_j}$ and $\mathfrak{B}_i \models \phi$, but $\mathfrak{B}_j \not\models \phi$.

Problem 1. *Consider* a set of agents $\mathcal{N} = \{1, \ldots, N\}$, and suppose that each agent $i \in \mathcal{N}$ is modeled as a tuple $\mathcal{M}_i = (\mathcal{T}_i, Sync_i, \Pi_i, L_i)$, and assigned a task in the form of an LTL formula ϕ_i over $\boldsymbol{\Pi}_i = \bigcup_{i' \in D_i} \Pi_{i'}$, for some $\{i\} \subseteq D_i \subseteq \mathcal{N}$. *For each* $i \in \mathcal{N}$ *find* a trace $\tau_i = s_{i,1}\alpha_{i,1}s_{i,2}\alpha_{i,2} \ldots$ of $\mathcal{T}_i$, and a synchronization sequence γ_i over $Sync_i$, with the property that regardless of the values of the transition time durations $\Delta_{\alpha_{i,1}}, \Delta_{\alpha_{i,2}} \ldots \in \mathbb{R}_0^+$, $i \in \mathcal{N}$, the set of the agents' behaviors $\{\beta_i = (\tau_i, \gamma_i, \mathbb{T}_i) \mid i \in \mathcal{N}\}$ is compatible, and $\mathfrak{B}_i = \{\beta_{i'} \mid i' \in D_i\}$ locally satisfies ϕ_i, for all $i \in \mathcal{N}$.

As each of the LTL formulas ϕ_i, $i \in \{1, \ldots, N\}$ over $\boldsymbol{\Pi}_i$ can be algorithmically translated into a deadlock-free language equivalent BA (Baier & Katoen, 2008), from now on, we pose the problem with the task specification of each agent i given as a BA $\mathcal{B}_i = (Q_i, q_{init,i}, \delta_i, \boldsymbol{\Sigma}_i = 2^{\boldsymbol{\Pi}_i}, F_i)$ and the local task satisfaction condition formulated as $w_{\mathfrak{B}_i} \in L(\mathcal{B}_i)$.

Remark 2. Collisions can be resolved either (i) through an LTL formula that forbids two agents to occupy the same cell of the environment or to exchange positions if they are in two neighboring cells, or (ii) through low-level controllers that implement the agents' transitions. This topic is however, beyond the scope of this paper.

A solution to Problem 1 can be obtained by imposing full synchronization and modifying the standard control plan synthesis procedure for TS from LTL specification (see, e.g., Kloetzer & Belta, 2008): (1) The set of agents is first partitioned into dependency classes similarly as in Guo and Dimarogonas (2015) by the iterative application of the rule that if $i' \in D_i$, then i' belongs to the same dependency class I_ℓ as i; (2) For each dependency class $I_\ell = \{1_\ell, \ldots, n_\ell\}$ and each agent $i \in I_\ell$, we set $\gamma_i = r_{i,1}r_{i,2} \ldots$, where $r_{i,j} = sync_i$. This step yields compatible behaviors of all agents in $\mathcal{N}$ independently of their traces and transition time durations. (3) For each $I_\ell = \{1_\ell, \ldots, n_\ell\}$, a TS $\mathcal{T}_\ell$ with $S_\ell = S_{1_\ell} \times \cdots \times S_{n_\ell}$ is constructed that represents the stepwise-synchronized traces of the agents within the class. (4) A product $\mathcal{P}_\ell$ of $\mathcal{T}_\ell$ and $\mathcal{B}_{1_\ell}, \ldots, \mathcal{B}_{n_\ell}$ is built that captures only the agents' traces that are admissible by $\mathcal{T}_\ell$ and result into behaviors that locally satisfy $\phi_{1_\ell}, \ldots, \phi_{n_\ell}$, respectively. The product $\mathcal{P}_\ell$ is analyzed using graph algorithms to find its accepting run that projects onto the desired traces of $\mathcal{T}_i$, for all $i \in I_\ell$. The outlined procedure is correct and complete. Although a certain level of decentralization is achieved via dependency partition, the algorithm suffers from a exponential growth of the product automaton state space with the increasing number of dependent agents, making the approach infeasible in practice. Furthermore, the solution requires that the agents synchronize after every single action execution, which potentially slows down the overall system performance.

In this work, we aim to reduce the computational demands of the above solution and to prevent the unnecessarily frequent synchronization. We propose to decompose the infinite horizon planning problem into an infinite sequence of finite horizon planning problems that are solved iteratively, upon the execution of the system. We build the dependency classes dynamically at each iteration. These classes are then often smaller than the offline ones, which has a dramatic impact on the efficiency of the planning procedure. We show that the stepwise synchronization scheme can be replaced with an event-triggered one. Finally, we prove, that under certain assumptions, the repetitive execution and recomputation of the plans leads to the satisfaction of all specifications.

4. Problem solution

This section provides the details of the proposed iterative solution to Problem 1. In Section 4.1, we set the preliminary synchronization sequences to be followed. In Section 4.2 we present a finite horizon plan synthesis algorithm that consists of four steps: (1) partitioning the agents into classes based on their dependency; and then for each of the classes separately: (2) building an intersection specification automaton up to a predefined horizon; (3) building a product capturing system behaviors admissible by the all agents within the class and by the intersection specification automaton up to a predefined horizon and evaluating the states of the product to reflect their respective profit towards the satisfaction of the specifications; and (4) finding and projecting a path in the product that leads to the most profitable state onto finite trace fragments of the individual agents. Section 4.3 discusses the iterative execution and recomputation.

Without loss of generality, let us assume that all agents in $\mathcal{N}$ form a single offline dependency class from Section 3.

4.1. Preliminary synchronization sequence

We set $\gamma_i = r_{i,1}r_{i,2}\ldots$, where $r_{i,j} = sync_i$, i.e., the preliminary synchronization sequence ensures full synchronization of all the agents after every single action execution. The behaviors of agents in $\mathcal{N}$ are thus compatible regardless of their traces and transition time durations. Namely, we have directly from Definitions 1 and 2:

Lemma 1. *Let $\gamma_i = r_{i,1}r_{i,2}\ldots$, where $r_{i,j} = sync_i$, for all $i \in \mathcal{N}$. For traces $\tau_i = s_{i,1}\alpha_{i,1}\ldots$, $\tau_{i'} = s_{i',1}\alpha_{i',1}\ldots$ of $\mathcal{T}_i, \mathcal{T}_{i'}$, $i, i' \in \mathcal{N}$, it holds that $t_{i,\alpha_j} = t_{i',\alpha_j}$, for all $j \geq 1$.*

4.2. Finite horizon planning

Besides the set of agents $\mathcal{N} = \{1, \ldots, N\}$ modeled as $\mathcal{M}_i = (\mathcal{T}_i, Sync_i, \Pi_i, L_i)$ and the specification BAs $\mathcal{B}_i = (Q_i, q_{init,i}, \delta_i, \boldsymbol{\Sigma}_i = 2^{\boldsymbol{\Pi}_i}, F)$, for all $i \in \mathcal{N}$, the inputs to the finite horizon planning algorithm are: the current states of $\mathcal{T}_1, \ldots, \mathcal{T}_N$, denoted $\mathfrak{s}_1, \ldots, \mathfrak{s}_N$, initially equal to $s_{init,1}, \ldots, s_{init,N}$, respectively; the current states of $\mathcal{B}_1, \ldots, \mathcal{B}_N$, denoted $\mathfrak{q}_1, \ldots, \mathfrak{q}_N$, initially equal to $q_{init,1}, \ldots, q_{init,N}$, respectively; a linear ordering $\prec$ over $\mathcal{N}$, initially arbitrary; fixed horizons $h, H \in \mathbb{N}$, which, loosely speaking, set the depth of planning in each BA and TS, respectively.

Dependency partitioning. We start with partitioning $\boldsymbol{\Phi} = \{\mathcal{B}_1, \ldots, \mathcal{B}_N\}$ into the smallest possible subsets $\Phi_1, \ldots, \Phi_M$, such that none of the transitions of any $\mathcal{B}_i \in \Phi_\ell$ that appears within the horizon h from the current state $\mathfrak{q}_i$ of $\mathcal{B}_i$ imposes restrictions on any agent i', where $\mathcal{B}_{i'} \notin \Phi_\ell$. This partition corresponds to the *necessary and sufficient dependency* up to the horizon h.

Definition 4 (*Participating Services*)**.** We call a set of services $\Pi_{i'}$, $i' \in D_i$ *participating* in $q \in Q_i$ if $i' = i$, or there exist $q' \in Q_i, \sigma \in \boldsymbol{\Sigma}_i$, and $\varsigma \subseteq \Pi_{i'}$ such that $(q, \sigma, q') \in \delta_i$, and $(q, (\sigma \setminus \Pi_{i'}) \cup \varsigma, q') \notin \delta_i$.

Given $q \in Q_i$, the *alphabet* $\boldsymbol{\Sigma}_i^h$ *of* $\mathcal{B}_i$ *up to the horizon* h is $\boldsymbol{\Sigma}_i^h(q) = 2^{\boldsymbol{\Pi}_i^h(q)}$, where

$$\boldsymbol{\Pi}_i^h(q) = \bigcup_{q' \in \hat{\delta}_i^j(q), 0 \leq j \leq h} \{\Pi_{i'} \mid \Pi_{i'} \text{ is participating in } q'\},$$

where $\hat{\delta}_i^j(q)$ denotes the set of states reachable from q in j steps (see Section 2).

Definition 5 (*Dependency Partition*)**.** Given that $\mathfrak{q}_1, \ldots, \mathfrak{q}_N$ are the respective current states of BAs $\mathcal{B}_1, \ldots, \mathcal{B}_N$, the *dependency partition* of $\boldsymbol{\Phi}$ is induced by the equivalence $\sim^h$ defined on $\boldsymbol{\Phi}$: (i) $\mathcal{B}_i \sim^h \mathcal{B}_i$, and (ii) if there exists $\mathcal{B}_j$, such that $\mathcal{B}_i \sim^h \mathcal{B}_j$, and $\Pi_{i'} \subseteq \boldsymbol{\Pi}_j^h(\mathfrak{q}_j)$ or $\Pi_j \subseteq \boldsymbol{\Pi}_{i'}^h(\mathfrak{q}_{i'})$, then also $\mathcal{B}_i \sim^h \mathcal{B}_{i'}$. The dependency partition of $\boldsymbol{\Phi}$ is then $\{\Phi_1, \ldots, \Phi_M\}$, where $(\mathcal{B}_i \sim^h \mathcal{B}_{i'}) \iff (\mathcal{B}_i \in \Phi_\ell \iff \mathcal{B}_{i'} \in \Phi_\ell)$. The dependency partition of the set of all agents $\mathcal{N}$ is $\mathcal{I} = \{I_1, \ldots, I_M\}$, where a *dependency class* I_ℓ, is such that $i \in I_\ell \iff \mathcal{B}_i \in \Phi_\ell$, for all $\ell \in \{1, \ldots, M\}$.

The dependency partition is the function of the current states $\mathfrak{q}_1, \ldots, \mathfrak{q}_N$. Hence it is dynamically recomputed in each iteration of our solution. Although the local satisfaction of ϕ_i for $i \in I_\ell$ depends on the traces of all the agents $j \in D_i$, it is not influenced by the agents $j \in D_i \setminus I_\ell$ within the horizon h. Thus, when we now concentrate on planning within the horizon h, we can safely treat each dependency class $I_\ell = \{1_\ell, \ldots, n_\ell\}$ separately. In the worst case, the dynamic dependency partition is equal to the offline one from Section 3. This is however, often not the case and even splitting a single offline dependency class into two online ones triggers an exponential improvement in terms of computational demands of further steps. For an example, see Section 6.

Intersection automaton. For each dependency class I_ℓ, we construct a finite automaton that represents the language intersection of the BAs in $\Phi_\ell = \{\mathcal{B}_{1_\ell}, \ldots, \mathcal{B}_{n_\ell}\}$ up to the pre-defined horizon h. In this step, we rely on the fact that the preliminary synchronization sequence set in Section 4.1 guarantees step-by-step synchronization. We label the states of the intersection automaton with values that indicate the progress towards the satisfaction of the desired tasks. Later on, these values are used to set temporary goals in the finite horizon plan synthesis.

Without loss of generality, let the automata in Φ_ℓ be ordered according to $\prec$, i.e. $i_\ell \prec j_\ell$, for all $1 \leq i < j \leq n$.

Definition 6 (*Intersection Automaton*)**.** The intersection automaton of $\mathcal{B}_{1_\ell}, \ldots, \mathcal{B}_{n_\ell}$ up to the horizon h is $\mathcal{A}^h = (Q_\mathcal{A}, q_{init,\mathcal{A}}, \boldsymbol{\Sigma}_\mathcal{A}, \delta_\mathcal{A}, F_\mathcal{A})$, where $Q_\mathcal{A} \subset Q_{1_\ell} \times \cdots \times Q_{n_\ell} \times \mathbb{N}$ is a finite set of states, generated as described below; $q_{init,\mathcal{A}} = (\mathfrak{q}_{1_\ell}, \ldots, \mathfrak{q}_{n_\ell}, 1)$; $\boldsymbol{\Sigma}_\mathcal{A} = \left\{\bigcup_{i_\ell \in I_\ell} \sigma_{i_\ell} \mid \sigma_{i_\ell} \in (2^{\Pi_{i_\ell}} \cup 2^{\mathcal{E}_{i_\ell}})\right\}$; Let $Q_\mathcal{A}^0 = \{q_{init,\mathcal{A}}\}$. For all $1 \leq j \leq h$, we define $(q'_{1_\ell}, \ldots, q'_{n_\ell}, k') \in Q_\mathcal{A}^j$ and $\big((q_{1_\ell}, \ldots, q_{n_\ell}, k), \sigma, (q'_{1_\ell}, \ldots, q'_{n_\ell}, k')\big) \in \delta_\mathcal{A}^j$ iff (i) $(q_{1_\ell}, \ldots, q_{n_\ell}, k) \in Q_\mathcal{A}^{j-1}$, (ii) for all $i_\ell \in I_\ell$, either $(q_{i_\ell}, \sigma \cap \boldsymbol{\Pi}_{i_\ell}, q'_{i_\ell}) \in \delta_{i_\ell}$, or $q_{i_\ell} = q'_{i_\ell}$, and $\varepsilon_{i_\ell} \in \sigma$, and (iii) $k' = k + 1$ if $q_{\kappa_\ell} \in F_{\kappa_\ell}$, where $\kappa = k \bmod n_\ell$ and $k' = k$ otherwise. Finally, $Q_\mathcal{A} = \bigcup_{0 \leq j \leq h} Q_\mathcal{A}^j$ and $\delta_\mathcal{A} = \bigcup_{1 \leq j \leq h} \delta_\mathcal{A}^j$; $F_\mathcal{A} = \{(q_{1_\ell}, \ldots, q_{n_\ell}, k) \in Q_\mathcal{A} \setminus \{q_{init,\mathcal{A}}\} \mid q_{\kappa_\ell} \in F_{\kappa_\ell}$, where $\kappa = k \bmod n_\ell\}$.

The intersection automaton is not interpreted over infinite words and hence, it is not a BA. However, it is an automaton and as such, it can be viewed as a graph (see Section 2). A path in $\mathcal{A}^h$ from the initial state $(\mathfrak{q}_{1_\ell}, \ldots, \mathfrak{q}_{n_\ell}, 1)$ to a state $(q_{1_\ell}, \ldots, q_{n_\ell}, k)$ corresponds to a path from $\mathfrak{q}_{i_\ell}$ to q_{i_ℓ} in each $\mathcal{B}_{i_\ell}$ and vice versa. Formally, these two lemmas follow from the construction:

Lemma 2. *Consider a path $q_1\sigma_1 q_2\sigma_2 \ldots \sigma_{m-1}q_m$ in $\mathcal{A}^h$, where q_j denotes the tuple $(q_{1_\ell,j}, \ldots, q_{n_\ell,j}, k_j) \in Q_\mathcal{A}$, for all $1 \leq j \leq m$. Let $\varpi_{i_\ell,j} = \sigma_j \cap (\Pi_{i_\ell} \cup \mathcal{E}_{i_\ell})$ denote the range restriction of σ_j to the services of agent i_ℓ, for all $1 \leq j \leq m-1$ and let $\varpi_{i_\ell,\iota_1} \ldots \varpi_{i_\ell,\iota_\mu}$ be the subsequence of non-silent elements of $\varpi_{i_\ell,1} \ldots \varpi_{i_\ell,m-1}$. Finally, let $\omega_{i_\ell,j} = \sigma_{\iota_j} \cap \boldsymbol{\Pi}_{i_\ell}$, for all $1 \leq j \leq \mu$. Then there exists a* correspondingrun *$\bar{\rho}_{i_\ell} = \bar{q}_{i_\ell,1} \ldots \bar{q}_{i_\ell,\mu+1} \ldots$ of $\mathcal{B}_{i_\ell}$ over each word $\bar{w}_{i_\ell} = \omega_{i_\ell,1} \ldots \omega_{i_\ell,\mu} \ldots$, with the property that (1) $q_{i_\ell,1} = \bar{q}_{i_\ell,1} = \mathfrak{q}_{i_\ell}$, (2) $q_{i_\ell,\iota_j+1} = \bar{q}_{i_\ell,j+1}$, for all $1 \leq j \leq \mu$, (3) $q_{i_\ell,j+1} = q_{i_\ell,j}$, for all $1 \leq j \leq m-1$, such that $j \neq \iota_{j'}$, for any $1 \leq j' \leq \mu$.*

Lemma 3. *Consider a run $\rho_{i_\ell} = q_{i_\ell,1}q_{i_\ell,2}\ldots$ of $\mathcal{B}_{i_\ell}$ over a word $w_{i_\ell} = \omega_{i_\ell,1}\omega_{i_\ell,2}\ldots \in (2^{\Pi_{i_\ell}})^\omega$, where $q_{i_\ell,1} = \mathfrak{q}_{i_\ell}$. Let $\sigma_1\sigma_2\ldots$ be*

a word over $\boldsymbol{\Sigma}_{\mathcal{A}}$ with the property that there exists its subsequence $\sigma_{\iota_1}\sigma_{\iota_2}\ldots$, such that $\sigma_{\iota_\iota} \cap \boldsymbol{\Pi}_{i_\ell} = \omega_{i_\ell,j}$, for all $j \geq 1$, while $\sigma_j \cap (\Pi_{i_\ell} \cup \mathcal{E}_{i_\ell}) = \mathcal{E}_{i_\ell}$, for all $j \geq 1$, $j \neq \iota_{j'}$, for any $j' \geq 1$. Then there exists a path $\bar{q}_1\sigma_1\bar{q}_2\sigma_2\ldots\sigma_{h-1}\bar{q}_h$ in $\mathcal{A}^h$, where $\bar{q}_j$ denotes the tuple $(\bar{q}_{1_\ell,j},\ldots,\bar{q}_{n_\ell,j},k_j) \in Q_{\mathcal{A}}$, such that: (1) $\bar{q}_{i_\ell,1} = q_{i_\ell,1} = q_{i_\ell}$, (2) $\bar{q}_{i_\ell,\iota_j+1} = q_{i_\ell,j+1}$, for all $1 \leq \iota_j \leq h$, and (3) $\bar{q}_{i_\ell,j+1} = \bar{q}_{i_\ell,j}$, for all $1 \leq j, \iota_{j'} < h$, where $j \neq \iota_{j'}$.

Through k, we remember the progress towards the satisfaction of the individual specifications ordered according to $\prec$; for $k \leq n_\ell$, an accepting state is guaranteed to be present on a projected finite path of each $\mathcal{B}_{i_\ell}$, $1 \leq i \leq k$. For $k > n_\ell$, an accepting state is surely present on each projected run of each $\mathcal{B}_{i_\ell}$, at least $\lceil \frac{k}{n_\ell} \rceil$-times for all $1 \leq i \leq \kappa$, and at least $\lfloor \frac{k}{n_\ell} \rfloor$-times for all $\kappa < i \leq n_\ell$, where $\kappa = k \bmod n_\ell$. To be able to identify "profitable" actions/transitions of the agents w.r.t. Φ_ℓ, and thus also w.r.t. $\boldsymbol{\Phi}$, we assume that at least a state which represents a step towards the satisfaction of the highest-order specification $\mathcal{B}_{1_\ell}$ is present in $\mathcal{A}^h$. In Section 4.2 this fact will allow us to set short-term goals in TSs $\mathcal{T}_{1_\ell},\ldots,\mathcal{T}_{n_\ell}$. We discuss its relaxation in Section 5.

Assumption 1. Assume that $F_{\mathcal{A}}$ is not empty.

Definition 7 (*Automaton Progressive Function*)**.** The progressive function $V_{\mathcal{A}} : Q_{\mathcal{A}} \to \mathbb{N}_0 \times \mathbb{Z}_0^-$ is for a state $q = (q_{1_\ell},\ldots,q_{n_\ell},k)$ defined as: $V_{\mathcal{A}}(q) = \big(k, -\min_{q_f \in F_{\mathcal{A}}} \mathrm{dist}(q, q_f)\big)$.

The increasing value of $V_{\mathcal{A}}$ indicates a progress towards the satisfaction of the individual local specifications in Φ_ℓ, ordered according to $\prec$. No progress can be achieved from a state q, such that $V_{\mathcal{A}}(q) = (k, -\infty)$ within the horizon h, and hence, we remove such states from $\mathcal{A}^h$. From Assumption 1, $V_{\mathcal{A}}(q_{init,\mathcal{A}}) = (1, d)$, where $d \neq -\infty$.

Product system. The intersection automaton and its progressive function allow us to assess which service sets should be provided in order to maximize the progress towards the satisfaction of the specifications. The remaining step is to plan the transitions of the individual agents to reach states in which these services are available. We do so through the definition of a product system P^H. Besides the behaviors permitted by the task specifications, the product system captures the allowed behaviors (finite trace fragments) of the agents from I_ℓ up to the horizon H. Similarly as the states of A^h, the states of P^H are evaluated to indicate their progress towards the specifications satisfaction.

Definition 8 (*Product System*)**.** The product system up to the horizon H of the agent TSs $\mathcal{T}_{i_\ell}$, $i_\ell \in I_\ell$, and the intersection automaton $\mathcal{A}^h$ from Definition 6 is an automaton $\mathcal{P}^H = (Q_{\mathcal{P}}, q_{init,\mathcal{P}}, \boldsymbol{\Sigma}_{\mathcal{P}}, \delta_{\mathcal{P}})$, where $Q_{\mathcal{P}} \subset S_{1_\ell} \times \cdots \times S_{n_\ell} \times Q_{\mathcal{A}}$ is a finite set of states, generated as described below; $q_{init,\mathcal{P}} = (\mathfrak{s}_{1_\ell},\ldots,\mathfrak{s}_{n_\ell},q_{init,\mathcal{A}})$; $\boldsymbol{\Sigma}_{\mathcal{P}} = A_{1_\ell} \times \cdots \times A_{n_\ell} \times \boldsymbol{\Sigma}_{\mathcal{A}}$; Let $Q_{\mathcal{P}}^0 = \{q_{init,\mathcal{P}}\}$. For all $1 \leq j \leq H$, $q = (s_{1_\ell},\ldots,s_{n_\ell},q_{\mathcal{A}})$, $\sigma = (\alpha_{1_\ell},\ldots,\alpha_{n_\ell},\sigma_{\mathcal{A}})$, $q' = (s'_{1_\ell},\ldots,s'_{n_\ell},q'_{\mathcal{A}})$, we define that $q' \in Q_{\mathcal{P}}^j$ and $(q,\sigma,q') \in \delta_{\mathcal{P}}^j$ iff (i) $q \in Q_{\mathcal{P}}^{j-1}$, (ii) for all $i_\ell \in I_\ell$, it holds that $T(s_{i_\ell},\alpha_{i_\ell}) = s'_{i_\ell}$, and $\sigma_{\mathcal{A}} \cap (\Pi_{i_\ell} \cup \mathcal{E}_{i_\ell}) = L_{i_\ell}(\alpha_{i_\ell})$, and (iii) $(q_{\mathcal{A}},\sigma_{\mathcal{A}},q'_{\mathcal{A}}) \in \delta_{\mathcal{A}}$. Finally, $Q_{\mathcal{P}} = \bigcup_{0\leq j\leq H} Q_{\mathcal{P}}^j$ and $\delta_{\mathcal{P}} = \bigcup_{1\leq j\leq H} \delta_{\mathcal{P}}^j$.

The set of accepting states $F_{\mathcal{P}}$ is not significant for the further computations, hence we omit it from $\mathcal{P}^H$. Analogously as $\mathcal{A}^h$, $\mathcal{P}^H$ can be viewed as a graph (see Section 2). A path in $\mathcal{P}^H$ can be projected onto a finite trace prefix of each $\mathcal{T}_{i_\ell}$, a finite run prefix of $\mathcal{A}^h$ and further through Lemma 2 onto a finite run prefix of each BA $\mathcal{B}_{i_\ell}$, too.

Definition 9 (*Projection*)**.** Consider a path $\varrho = q_1\sigma_1q_2\ldots q_{m-1}\sigma_{m-1}q_m$ in $\mathcal{P}^H$, where $q_1 = q_{init,\mathcal{P}}$. ϱ can be projected onto a finite trace prefix $\hat{\tau}_{i_\ell}(\varrho)$ of each $\mathcal{T}_{i_\ell}$, $i_\ell \in I_\ell$ in the expected way: for all $1 \leq j \leq m$, the jth state and action of $\hat{\tau}_{i_\ell}(\varrho)$ is $s_{i_\ell,j}$ and $\alpha_{i_\ell,j}$ if the jth state and transition label of ϱ is $q_j = (s_{1_\ell,j},\ldots,s_{n_\ell,j},q_{\mathcal{A},j})$, and $\sigma_j = (\alpha_{1_\ell,j},\ldots,\alpha_{n_\ell,j},\sigma_{\mathcal{A},j})$, respectively. Furthermore, for all $1 \leq j \leq m$, the jth state of the projected run prefix $\hat{\rho}_{\mathcal{A}}(\varrho)$ of $\mathcal{A}^h$ is $q_{\mathcal{A},j}$ if the jth state of ϱ is $q_j = (s_{1_\ell},\ldots,s_{n_\ell},q_{\mathcal{A},j})$; and assuming that $q_{\mathcal{A},j} = (q_{1_\ell,j},\ldots,q_{n_\ell,j},k)$, the state of the corresponding state sequence $\hat{\rho}_{i_\ell}(\varrho)$ in $\mathcal{B}_{i_\ell}$ is $q_{i_\ell,j}$.

Although $\hat{\rho}_{i_\ell}(\varrho)$ is a projection of $q_{\mathcal{A},j}$, i.e., a sequence of states of $\mathcal{B}_{i_\ell}$, it might not be a run of $\mathcal{B}_{i_\ell}$ as such. However, thanks to Lemma 2, $\hat{\rho}_{i_\ell}(\varrho)$ maps to a unique corresponding run $\bar{\rho}_{i_\ell}(\varrho)$ of $q_{\mathcal{B}_{i_\ell}}$. Note that in what follows, we distinguish between $\hat{\rho}_{i_\ell}(\varrho)$ and $\bar{\rho}_{i_\ell}(\varrho)$.

Definition 10 (*Progressive Function and State*)**.** The progressive function $V_{\mathcal{P}} : Q_{\mathcal{P}} \to \mathbb{N}_0 \times \mathbb{Z}_0^-$ is inherited from the intersection automaton $\mathcal{A}^h$ (Definition 7), i.e., for all $q = (s_{1_\ell},\ldots,s_{n_\ell},q_{\mathcal{A}}) \in Q_{\mathcal{P}}$, $V_{\mathcal{P}}\big((s_{1_\ell},\ldots,s_{n_\ell},q_{\mathcal{A}})\big) = V_{\mathcal{A}}(q_{\mathcal{A}})$. A state $q \in Q_{\mathcal{P}}$ is a *progressive state* if $V_{\mathcal{P}}(q) > V_{\mathcal{P}}(q_{init,\mathcal{P}})$. A *maximally progressive state* is a progressive state q, with the property that for all $q' \in Q_{\mathcal{P}}$, it holds $V_{\mathcal{P}}(q) \geq V_{\mathcal{P}}(q')$.

Finite horizon plan synthesis. To find a suitable finite horizon plan, we impose the following assumption and discuss its relaxation in Section 5. It states that within H, at least one service set can be provided that makes at least one step towards the local satisfaction of the highest-priority formula ϕ_{1_ℓ}.

Assumption 2. Assume that there is a progressive state q_p reachable in $\mathcal{P}^H$ via a finite path $q_1\sigma_1\ldots\sigma_{m-1}q_m$, where $q_1 = q_{init,\mathcal{P}}$, $q_m = q_p$, and $L(\alpha_{1_\ell,j}) \neq \mathcal{E}_{1_\ell,j}$, for some $\sigma_j = (\alpha_{1_\ell,j},\ldots,\alpha_{n_\ell,j},\sigma_{\mathcal{A},j})$, $1 \leq j \leq m-1$.

We compute a suitable finite horizon plan as the shortest path $\varrho = q_1\sigma_1q_2\ldots q_m\sigma_mq_m$ in $\mathcal{P}^H$ from $q_1 = q_{init,\mathcal{P}}$ to a maximally progressive state $q_m = q_{max}$. Such a path can be found in linear time with respect to the size of $\mathcal{P}^H$ (see Section 2). The projection of ϱ onto the individual TSs gives finite trace fragments $\hat{\tau}_{i_\ell}(\varrho) = \mathfrak{s}_{i_\ell}\alpha_{i_\ell,1}s_{i_\ell,2}\ldots s_{i_\ell,m-1}\alpha_{i_\ell,m-1}s_{i_\ell,m}$, to be followed by each agent $i_\ell \in I_\ell$. Alg. 1 summarizes the proposed finite horizon planning for the set of all agents $\mathcal{N}$. Since this algorithm will be run iteratively, starting from its second execution on, it also takes as an input the fragments $\hat{\tau}_1,\ldots,\hat{\tau}_N$, $\hat{\rho}_1,\ldots,\hat{\rho}_N$ and the progressive function value of the maximally progressive state from the previous iteration. In comparison to the solution from Section 3, we treat separately each dynamic dependency class (lines 2–13). These classes are often smaller than the offline dependency classes, thereby increasing the level of decentralization of the planning procedure.

4.3. Infinite horizon replanning

To complete the solution, we discuss the infinite, iterative execution and recomputation of the finite horizon plans. We present two recomputation strategies: a *stepwise* one, and later on in Section 4.3.2 an *event-triggered* one, where we adjust the preliminary synchronization sequences to reduce the synchronization frequency.

4.3.1. Stepwise solution

In each iteration of the stepwise solution, the finite horizon plans are executed as summarized in Alg. 2. Each agent $i \in \mathcal{N}$ first sets its preliminary synchronization sequence $\gamma_i = r_{i,1}r_{i,2}\ldots$

Algorithm 1 Procedure short_horizon_plan

Input: a set of agents $\mathcal{N} = \{1, \ldots, N\}$, their models $\mathcal{M}_1, \ldots, \mathcal{M}_N$; BAs $\mathcal{B}_1, \ldots, \mathcal{B}_N$; current states $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$, $\mathfrak{q}_1, \ldots, \mathfrak{q}_n$; an ordering $\prec$ over $\mathcal{N}$; and planning horizons $h, H \in \mathbb{N}$, previous fragments $\hat{\tau}_1, \ldots, \hat{\tau}_N$, $\hat{\rho}_1, \ldots, \hat{\rho}_N$, previous maximal progressive function value $V_{\mathcal{P},max}$

Output: finite trace fragments $\hat{\tau}_1, \ldots, \hat{\tau}_N$ of $\mathcal{T}_1, \ldots, \mathcal{T}_N$; finite state sequences $\hat{\rho}_1, \ldots, \hat{\rho}_N$ of $\mathcal{B}_1, \ldots, \mathcal{B}_\mathcal{N}$; and a maximal progressive function value $V_{\mathcal{P},max}$

1: compute the partition $\mathcal{I} = \{I_1, \ldots, I_M\}$ (Def. 5)
2: **for all** $\ell \in \{1, \ldots, M\}$ **do**
3: construct $\mathcal{A}^h$ (Def. 6) and construct $\mathcal{P}^H$ (Def. 8)
4: find a maximally progressive state q_{max} in $\mathcal{P}^H$
5: **if** $V_\mathcal{P}(q_{max}) > V_{\mathcal{P},max}$ **then**
6: find the shortest path ϱ to q_{max} and update $V_{\mathcal{P},max}$
7: **for all** $i_\ell \in I_\ell$ **do**
8: $\hat{\tau}_{i_\ell} := \hat{\tau}_{i_\ell}(\varrho)$ and $\hat{\rho}_{i_\ell} := \hat{\rho}_{i_\ell}(\varrho)$ (Def. 9)
9: **end for**
10: **else**
11: remain $\hat{\tau}_1, \ldots, \hat{\tau}_N, \hat{\rho}_1, \ldots, \hat{\rho}_N, V_{\mathcal{P},max}$ unchanged
12: **end if**
13: **end for**
14: **return** $\hat{\tau}_1, \ldots, \hat{\tau}_N, \hat{\rho}_1, \ldots, \hat{\rho}_N, V_{\mathcal{P},max}$

and synchronizes with the other agents, i.e., sends $r_{i,1} = sync_i$, and waits for the reception of $sync_{i'}$ from all $i' \in \mathcal{N}$. Second, it computes the finite horizon plans by Alg. 1. Third, it executes the first action $\alpha_{i,1}$ of the planned trace fragment $\hat{\tau}_i = \mathfrak{s}_i\alpha_{i,1}s_{i,2}\ldots s_{i,m-1}\alpha_{i,m-1}s_{i,m}$ along with the first silent or non-silent service set $L(\alpha_{i,1})$. It transitions to state $s_{i,2}$ and the current state $\mathfrak{s}_i$ is updated. The previous synchronization through $r_{i,1}$ has ensured that each agent $i' \in I_\ell$ executes its respective action $\alpha_{i',1}$ simultaneously. At the same time, the current state of the BAs $\mathcal{B}_i$, is updated to the second state $q_{i,2}$ of the run fragment $\hat{\rho}_i$. Note, that if $L(\alpha_{i,1}) = \varepsilon_i$, then $q_{i,2} = q_{i,1}$ by Definitions 6 and 8. Furthermore, if $q_{i,2} \in F_i$, then we update the ordering $\prec$ so that i becomes of the lowest priority (the highest order), while maintaining the mutual ordering of all the other agents. This change reflects that a step towards the local satisfaction of $\mathcal{B}_i$ has been made and in the next iteration, progressing towards another agent's specification will be prioritized. Finally, all the agents start another iteration of the algorithm simultaneously as prescribed by the synchronization sequence. For the simplicity of the presentation, we assumed that the computation of short_horizon_plan does not take any time. In practice, we would cope with different computation times via synchronization both before and after running the procedure short_horizon_plan.

The motivation for replanning after every iteration on line 11 of Alg. 2 is to gradually shift the planning horizon in order to keep the planning procedure sufficiently informed about the future possibilities. Less frequent replanning may be more efficient in terms of computational demands, at the cost of inefficiency of the resulting plans. Although the TSs and the BAs are finite and hence there are a finite number of different finite horizon problems to solve, the considered LTL formulas are interpreted over infinite time and the number of iterations cannot be generally upper-bounded due to the arbitrary transition durations. An alternative to the infinite number of executions would be to remember how the finite horizon problem has been resolved for every single combination of the TSs and BAs states; however, this is generally not feasible due to the fact that the number of these combinations grows quickly with the number of agents, the size of the environment and the complexity of the tasks.

4.3.2. Event-triggered solution

In the stepwise solution, the synchronization takes place after every single transition of every agent, which might be more frequently than necessarily needed. For example, if the first m actions

Algorithm 2 Stepwise solution to Prob. 1

Input: a set of agents $\mathcal{N} = \{1, \ldots, N\}$, their models $\mathcal{M}_1, \ldots, \mathcal{M}_N$; BAs $\mathcal{B}_1, \ldots, \mathcal{B}_N$; and horizons $h, H \in \mathbb{N}$

Output: traces $\tau_1, \ldots, \tau_N$ and synchronization sequences $\gamma_1, \ldots, \gamma_N$ that are a solution to Problem 1, and sequences $\rho_1, \ldots, \rho_n$ of states of BAs $\mathcal{B}_1, \ldots, \mathcal{B}_N$.

1: **for all** $i \in \mathcal{N}$ **do**
2: initialize $\prec := (1, \ldots, N)$; $\mathfrak{s}_i := s_{init,i}$ $\mathfrak{q}_i := q_{init,i}$
3: initialize $\hat{\tau}_i :=$ empty; $\hat{\rho}_i :=$ empty; $V_{\mathcal{P},max} := (0, 0)$
4: initialize synchronization sequence γ_i (Sec. 4.1)
5: send $r_{i,1} := sync_i$ and wait for $sync_{i'}$ from all $i' \in \mathcal{N}$
6: $j_i := 2$
7: **end for**
8: **while** true **do**
9: $\hat{\tau}_1, \ldots, \hat{\tau}_N, \hat{\rho}_1, \ldots, \hat{\rho}_N, V_{\mathcal{P},max}$:= short_horizon_plan
10: **for all** $i \in \mathcal{N}$ **do**
11: execute $\alpha_{i,1}$, provide $L(\alpha_{i,1})$; $\mathfrak{s}_i := s_{i,2}$; $\mathfrak{q}_i := q_{i,2}$
12: **if** $\mathfrak{q}_i \in F_i$ **then**
13: reorder $\prec$, s.t. $i' \prec i$, for all $i' \in \{1, \ldots, N\} \setminus \{i\}$
14: $V_{\mathcal{P},max} := (0, 0)$
15: **end if**
16: send $r_{i,j_i} := sync_i$, wait for $sync_{i'}$ from all $i' \in \mathcal{N}$
17: $j_i := j_i + 1$
18: **end for**
19: **end while**
20: **return** $\tau_1, \ldots, \tau_N, \gamma_1, \ldots, \gamma_n, \rho_1, \ldots, \rho_n$

in the planned trace fragment $\hat{\tau}_i$ of an agent i are all associated with silent services, then this agent does not need to synchronize with the others nor to recompute its planned trace fragment. In what follows, we adapt the preliminary synchronization sequence so that the synchronization and recomputation are triggered by the need of collaboration.

Assume that for agent i the procedure short_horizon_plan executed on line 9 of Alg. 2 has returned a trace and a run fragment denoted by $\hat{\tau}_i = s_{i,1}\alpha_{i,1}s_{i,2}\alpha_{i,2}\ldots s_{i,m}$ and $\hat{\rho}_i = q_{i,1}q_{i,2}\ldots q_{i,m}$, for some $m \geq 2$. The main idea is to postpone the synchronization with the others from after the execution of $\alpha_{i,1}$ till the time $t_{s_{i,j}}$ right before the execution of the action $\alpha_{i,j}, j > 1$ with one of the following properties: $L(\alpha_{i,j})$ is non-silent, or $q_{i,j}$ is accepting, or $q_{i,j} = q_{i,m}$, or there exists an agent i', such that $sync_{i'}$ has been received during $(t_{\alpha_{i,j-1}}, t_{s_{i,j}}]$. If one of the conditions is met, the agent sends $sync_i$, and waits for receiving $sync_{i'}$, from all $i' \in \mathcal{N}$. The finite horizon plans are recomputed and the event-triggered recomputation procedure repeats. On the other hand, if none of the conditions is met, the agent i substitutes the synchronization action $sync_i$ planned within the preliminary synchronization sequence γ_i with $nosync_i$. Note that thanks to the enforced compatibility of the behaviors, a deadlock is prevented. The solution is summarized in Alg. 3.

Remark 3. In Remark 1, we introduced parametrized synchronization requests. Assume that $\mathcal{I} = \{I_1, \ldots, I_M\}$ is an offline dependency partition from Section 3. Then for each agent $i \in I_\ell$, we can replace $sync_i$ with $sync_i(I_\ell)$, and every $nosync_i$ with $sync_i(\{i\})$. However, an analogous step cannot be applied in the case of dynamic classes.

5. Solution analysis and discussion

Lemma 4. *Let $\tau_1, \ldots, \tau_N, \gamma_1, \ldots, \gamma_n, \rho_1, \ldots, \rho_n$ be the traces, the synchronization sequences and the corresponding state sequences returned by Alg. 2. Then ρ_i contains infinitely many states $f_i \in F_i$, for all $i \in \mathcal{N}$.*

Proof. Denote $\tau_i = s_{i,1}s_{i,2}\ldots$, $\rho_i = q_{i,1}q_{i,2}\ldots$, and $\mathbb{T}(\tau_i) = t_{s_{i,1}}t_{s_{i,2}}\ldots$, for all $i \in \mathcal{N}$. Consider a time instance $t_{s_{i,j}}$, where $j \geq 1$ and assume that i is the most prioritized agent at $t_{s_{i,j}}$,

Algorithm 3 Event-triggered solution to Prob. 1

Input: a set of agents $\mathcal{N} = \{1, \ldots, N\}$, their models $\mathcal{M}_1, \ldots, \mathcal{M}_N$; BAs $\mathcal{B}_1, \ldots, \mathcal{B}_N$; and horizons $h, H \in \mathbb{N}$

Output: traces $\tau_1, \ldots, \tau_N$ and synchronization sequences $\gamma_1, \ldots, \gamma_N$ that are a solution to Problem 1, and sequences $\rho_1, \ldots, \rho_n$ of states of BAs $\mathcal{B}_1, \ldots, \mathcal{B}_N$.

1: **for all** $i \in \mathcal{N}$ **do**
2: initialize $\prec := (1, \ldots, N)$; $\mathfrak{s}_i := s_{init,i}$ $\mathfrak{q}_i := q_{init,i}$
3: initialize $\hat{\tau}_i :=$ empty; $\hat{\rho}_i :=$ empty; $V_{\mathcal{P},max} := (0, 0)$
4: initialize synchronization sequence γ_i (Sec. 4.1)
5: send $r_{i,1} := sync_i$ and wait for $sync_{i'}$ from all $i' \in \mathcal{N}$
6: $j_i := 2$
7: **end for**
8: **while** true **do**
9: $\hat{\tau}_1, \ldots, \hat{\tau}_N, \hat{\rho}_1, \ldots, \hat{\rho}_N :=$ short_horizon_plan
10: **for all** $i \in \mathcal{N}$ **do**
11: execute $\alpha_{i,1}$ and provide service set $L(\alpha_{i,1})$
12: $\mathfrak{s}_i := s_{i,2}$; $\mathfrak{q}_i := q_{i,2}$; $k_i := 2$
13: **while** $L(\alpha_{i,j_i}) = \varepsilon_i$ and $\mathfrak{q}_i \notin F_i$ and $\mathfrak{q}_i$ is not the last element of $\hat{\rho}_i$ and $sync_{i'}$ was not received from any $i' \in \mathcal{N} \setminus \{i\}$ during the last iteration **do**
14: send $r_{i,j_i} := nosync_i$; $j_i := j_i + 1$
15: execute α_{i,k_i} and provide service set $L(\alpha_{i,k_i})$
16: $\mathfrak{s}_i := s_{i,k_i+1}$; $\mathfrak{q}_i := q_{i,k_i+1}$; $k_i := k_i + 1$
17: **end while**
18: **if** $\mathfrak{q}_i \in F_i$ **then**
19: reorder $\prec$, s.t. $j \prec i$, for all $j \in \{1, \ldots, N\} \setminus \{i\}$
20: $V_{\mathcal{P},max} := (0, 0)$
21: **end if**
22: send $r_{i,j_i} := sync_i$, wait for $sync_{i'}$ from all $i' \in \mathcal{N}$
23: $j_i := j_i + 1$
24: **end for**
25: **end while**
26: **return** $\tau_1, \ldots, \tau_N, \gamma_1, \ldots, \gamma_n, \rho_1, \ldots, \rho_n$

i.e., that $i \prec i'$, for all $i' \in \mathcal{N}$. Let ϱ be the path to a maximally progressive state q_{max} of $\mathcal{P}^H$ at time $t_{s_{i,j}}$ computed on line 4 of Alg. 1 and let $\hat{\tau}_i$ and $\hat{\rho}_i$ be the corresponding finite trace prefix of $\mathcal{T}_i$, and the state sequence of $\mathcal{B}_i$ computed on lines 7–9. Assume for a moment that the dependency partition $\mathcal{I}$ remains the same at time $t_{s_{i,j+1}}$. Regardless of the steps taken in Alg. 2, the state q_{max} is also present in $\mathcal{P}^H$ at time $t_{s_{i,j+1}}$, hence if condition on line 5 is not satisfied, this state remains the progressive goal. Now, let ϱ' be the path to a maximally progressive state q'_{max} of $\mathcal{P}^H$ at time $t_{s_{i,j+1}}$ and let $\hat{\tau}_i'$ and $\hat{\rho}_i'$ be the corresponding finite trace prefix of $\mathcal{T}_i$, and the state sequence of $\mathcal{B}_i$. If $q'_{max} \neq q_{max}$, then $V_{\mathcal{P}}(q'_{max}) > V_{\mathcal{P}}(q_{max})$ and q'_{max} is, loosely speaking, closer to reaching an accepting state of $\mathcal{B}_i$. Thanks to Assumption 1 and Assumption 2 and the finite number of states of $\mathcal{P}^H$, by repetitive reasoning we get that there exists time $t_{s_{i,m}}$, when a state q^* of $\mathcal{P}_H$ is reached, such that $V_{\mathcal{P}}(q^*) \geq V_{\mathcal{P}}(q_{max})$. Inductively, a state q_f of P^H that projects onto an accepting state of $\mathcal{B}_i$ will eventually be reached. Similar holds even if the dependency partition $\mathcal{I}$ has changed at time $t_{s_{i,j+1}}$ and i is present in a dependency class $I_{\ell'} \neq I_\ell$. The state $q_{max} = (s_{1_\ell,m}, \ldots, s_{n_\ell,m}, (q_{1_\ell,m}, \ldots, q_{n_\ell,m}, k_m))$ can be mapped onto a state of the new $\mathcal{P}^H$ at time $t_{s_{i,j+1}}$, i.e., onto $q_{max,\ell'} = (s_{1_{\ell'},m'}, \ldots, s_{n_{\ell'},m'}, (q_{1_{\ell'},m'}, \ldots, q_{n_{\ell'},m'}, k_{m'}))$, where $s_{i',m} = s_{i',m'}$, and $q_{i',m} = q_{i',m'}$, for all $i' \in I_\ell \cap I_{\ell'}$. The remainder of the proof is analogous to the above. Finally, it is ensured that at least one non-silent service is provided by $\mathcal{T}_i$ on the executed path to q_f. Lines 12–15 of Alg. 2 ensure that each $i \in \mathcal{N}$ will repeatedly become the most prioritized one. Altogether, ϕ_i contains infinitely many states $f_i \in F_i$, for all $i \in \mathcal{N}$.

In summary, Lemma 1 gives us the existence of compatible behaviors regardless of the traces of the agent TSs and the transition time durations. Lemma 4 together with Lemmas 2 and 3, and Definition 8 yield that $\tau_1, \ldots, \tau_N$ produce words that are accepted by each $\mathcal{B}_i$.

Theorem 1. *The traces $\tau_1, \ldots, \tau_N$ together with the synchronization sequences $\gamma_1, \ldots, \gamma_N$ returned by Alg. 2 provide a solution to Problem 1.*

To prove the correctness of the event-triggered solution, we have to prove that the computed traces and synchronization sequences (i) yield compatible behaviors and (ii) locally satisfy the LTL formulas:

Lemma 5. *The traces $\tau_1, \ldots, \tau_N$, where $\tau_i = s_{i,1}\alpha_{i,1} \ldots$, for all $i \in \mathcal{N}$, together with the synchronization sequences $\gamma_1, \ldots, \gamma_N$ returned by Alg. 3 yield compatible behaviors regardless of the values of the transition time durations $\Delta_{\alpha_{i,1}}, \Delta_{\alpha_{i,2}} \ldots$.*

Proof. Follows immediately from the condition of the while loop on line 13 of Alg. 3.

Lemma 6. *Let $\tau_1, \ldots, \tau_N, \gamma_1, \ldots, \gamma_N, \rho_1, \ldots, \rho_N$ be the traces, the synchronization sequences and the corresponding state sequences returned by Alg. 3. Then ρ_i contains infinitely many states $f_i \in F_i$, for all $i \in \mathcal{N}$.*

Proof. Denote $\tau_i = s_{i,1}\alpha_{i,1} \ldots$, $\rho_i = q_{i,1}q_{i,2} \ldots$, and $\mathbb{T}_i = t_{s_{i,1}}t_{\alpha_{i,1}}t_{s_{i,2}}t_{\alpha_{i,2}} \ldots$, for all $i \in \mathcal{N}$. To prove the lemma, we prove that at each time t, it holds that an accepting state $f_i \in F_i$ will be eventually reached for the most prioritized agent i at time t. First, consider $t = 0$. Without loss of generality, assume that $i \prec i'$, for all $i' \in \mathcal{N}$. Denote $I_\ell \in \mathcal{I}$ the dependency class i belongs to. Let $\hat{\tau}_1, \ldots, \hat{\tau}_N$ and $\hat{\rho}_1, \ldots, \hat{\rho}_N$ be the finite trace prefixes of $\mathcal{T}_1, \ldots, \mathcal{T}_N$, and the state sequences of $\mathcal{B}_1, \ldots, \mathcal{B}_N$ computed on line 9 of Alg. 3, respectively. Specifically for I_ℓ, these were obtained from the projections of the shortest path ϱ_ℓ that leads to a maximally progressive state $q_{max,\ell} = (s_{1_\ell,m} \ldots s_{n_\ell,m}, (q_{1_\ell,m}, \ldots, q_{n_\ell,m}, k_m))$ of the product system $\mathcal{P}^H_\ell$ computed on line 4 of Alg. 1. Let the execution proceed as described in Alg. 3 and let $\mathfrak{t}_s$ denote the first time instance after $t = 0$ with one of the properties triggering the synchronization, i.e., with one of the conditions of line 13 of Alg. 3 being false. Note that $\mathfrak{t}_s$ is finite and after the requested synchronization is performed, the time $\mathfrak{t}_\alpha$ is equal to some $t_{s_{i',j}}$ in $\mathbb{T}_{i'}$, for all $i' \in \mathcal{N}$. Let $\mathfrak{s}_{i'}$ and $\mathfrak{q}_{i'}$ denote the respective states of $\mathcal{T}_{i'}$ and $\mathcal{B}_{i'}$ at time $\mathfrak{t}_\alpha$, and let $\vec{\tau}_{i'}$ and $\vec{\rho}_{i'}$ denote the respective suffixes of $\hat{\tau}_{i'}$ and $\hat{\rho}_{i'}$ starting at $\mathfrak{s}_{i'}$ and $\mathfrak{q}_{i'}$, for all $i' \in \mathcal{N}$. Denote by $I_{\ell'}$ the dependency class i belongs to at time $\mathfrak{t}_\alpha$ and by $\mathcal{P}^H_{\ell'}$ the corresponding product system. We now show the existence of a mapping between the suffixes $\vec{\tau}_{i'}$ and $\vec{\rho}_{i'}$ of agents $i' \in I_{\ell'} \cap I_\ell$ onto a single finite path $\varrho_{\ell'}$ in $\mathcal{P}^H_{\ell'}$, whose length is strictly smaller than the length of ϱ_ℓ in $\mathcal{P}^H_\ell$. Trivially, if $s_{i',m} = \mathfrak{s}_{i'}$, for all $i' \in I_{\ell'} \cap I_\ell$, the path $\varrho_{\ell'}$ is empty. Suppose that $s_{i',m} \neq \mathfrak{s}_{i'}$, for some $i' \in I_{\ell'} \cap I_\ell$. We propose $\varrho_{\ell'}$ to be the path whose projections onto the states of $\mathcal{T}_{i'}$ and $\mathcal{B}_{i'}$ are the respective sequences $\mathfrak{s}_{i'} \ldots \mathfrak{s}_{i'}\vec{\tau}_{i'}$ and $\mathfrak{q}_{i'} \ldots \mathfrak{q}_{i'}\vec{\rho}_{i'}$, such that the length of these two sequences, and hence also the length of $\varrho_{\ell'}$, is equal to the length of the suffix $\vec{\tau}_{i''}$ that is the longest one among the agents $i'' \in I_{\ell'} \cap I_\ell$. This path is strictly shorter than ϱ_ℓ. At the same time, it exists in $\mathcal{P}^H_{\ell'}$ due to the assumption that $s \xrightarrow{\alpha} s$, for all $s \in S_{i'}$, $i' \in \mathcal{N}$, and some $\alpha \in A_{i'}$. The existence of $\varrho_{\ell'}$ ensures that a progress towards some $f_i \in F_i$. Thanks to Assumptions 1 and 2 and the finite number of states of the product system, by repetitive reasoning we get that during the execution of Alg. 3, since a certain moment on, the last state of ϱ'_ℓ will be the maximally progressive state $q_{max,\ell'}$ in $\mathcal{P}^H_{\ell'}$, until this state is reached. Inductively, a state q_f that projects onto an accepting state of $\mathcal{B}_i$ will eventually be reached. At the same time, at least one non-silent service is provided by $\mathcal{T}_i$ on the path to q_f. Furthermore, lines 18–21 of Alg. 3 ensure, that each $i \in \mathcal{N}$ will repeatedly become the most prioritized one. Altogether, ρ_i contains infinitely many states $f_i \in F_i$, for all $i \in \mathcal{N}$.

Theorem 2. *The traces $\tau_1, \ldots, \tau_N$ together with the synchronization sequences $\gamma_1, \ldots, \gamma_N$ returned by Alg. 2 provide a solution to Problem 1.*

The complexity of one iteration of the solution is linear with respect to the size of $\mathcal{P}^H$, as the applied graph search algorithms are linear (see, e.g. Cormen et al., 2001). The size of the intersection automaton $\mathcal{A}^h$ is in $\mathcal{O}(n^{|I_\ell|+1})$, where n is the maximal set of states reachable in some $\mathcal{B}_i$, $i \in I_\ell$ within horizon h. The size of the product $\mathcal{P}^H$ is $\mathcal{O}(n^{|I_\ell|+1})$, where n is the maximal set of states reachable in some $\mathcal{B}_i$ or $\mathcal{T}_i$, $i \in I_\ell$ within the horizon H. In the worst case, when n reaches the sizes of $\mathcal{B}_i$ or $\mathcal{T}_i$ respectively, and when the number of dependency classes $n_\ell = 1$, the complexity of one iteration reaches the one of the straightforward solution, i.e., the complexity of one iteration is in $\mathcal{O}(N \cdot \prod_{i\in N} |\mathcal{T}_i| \cdot |\mathcal{B}_i|)$, where $|\mathcal{T}_i|$ and $|\mathcal{B}_i|$ is the size of $\mathcal{T}_i$ and $\mathcal{B}_i$, respectively. However, as we demonstrate in Section 6, a dramatic improvement of computational times can be achieved in practice.

Assumption 1 may be violated for two different reasons: First, if the selected horizon h is too short, and although $F_{\mathcal{A}} = \emptyset$ in $\mathcal{A}^h$, there exists $h' > h$, such that $F_{\mathcal{A}} \neq \emptyset$ in $\mathcal{A}^{h'}$. Second, if $F_{\mathcal{A}} = \emptyset$ even for $h \to \infty$. We propose to systematically extend the horizon h and update the automaton $\mathcal{A}^h$ until a set of states $F_{\mathcal{A}}$ becomes nonempty, or until the extension does not change the automaton $\mathcal{A}^h$ any more. In the former case, the automaton $\mathcal{A}^h$ with the extended horizon satisfies Assumption 1 and thus is used in constructing $\mathcal{P}^H$, maintaining the remainder of the solution as described in Sections 4.2 and 4.3. In the latter case the specification has become infeasible, indicating that a wrong step has been made in past. Therefore, we backtrack along the executed solution to a point when another service could have been executed instead of the one that has been already done. Intuitively, we "undo" the service and mark this service as forbidden in the specification automata. The backtracking procedure is roughly summarized in Alg. 4. In order to perform the backtracking, the system execution prefixes have to be remembered. To reduce the memory requirements, note that cycles between two exact same system execution states can be removed from the system execution prefixes without any harm. As there are only finitely many transitions possible in each system state of each TS and each BA, the backtracking procedure will ensure that eventually, the agents' trace prefixes will be found by Alg. 1 without further backtracking. Once Assumption 1 holds, there is only one reason for violation of Assumption 2, which is that the planning horizon H is not long enough. To cope with this, we systematically extend the horizon H similarly as we extended h in the BA. Eventually, a progressive state will be found.

Algorithm 4 Backtracking

Input: Transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_N$; BAs $\mathcal{B}_1, \ldots, \mathcal{B}_N$; *System execution prefix* $(\tau_1^t, \ldots, \tau_N^t, \rho_1^t, \ldots, \rho_N^t)$ up to the current time t, where $\tau_i^t = s_{i,1}\varpi_{i,1} \ldots \varpi_{i,t-1}s_{i,t}$, and $\rho_i^t = \rho_{i,1} \ldots \rho_{i,t}$, for all $i \in \{1, \ldots N\}$.
Output: Updates to BAs $\mathcal{B}_1, \ldots, \mathcal{B}_N$

1: $k := t$
2: **while** solution not found **do**
3: $k := k - 1$
4: Check, if the execution of $\bigcup_{1\in\{1,\ldots,N\}} \varpi_{i,k}$ can lead to a different set of states of BAs than to $q_{1,t}, \ldots, q_{N,t}$. If so, apply the change and goto line 6.
5: Forbid $\bigcup_{1\in\{1,\ldots,N\}} \varpi_{i,k}$ in the states $q_{1,k}, \ldots, q_{N,k}$ of each respective automaton $\mathcal{B}_1, \ldots, \mathcal{B}_N$
6: Execute one iteration of Alg. 1 from $s_1 = s_{1,k}, \ldots, s_N = s_{N,k}, q_1 = q_{1,k}, \ldots, q_N = q_{N,k}$
7: If a plan was found in line 5, continue with execution of Alg. 1, otherwise goto line 2 of Backtracking.
8: **end while**

Note, that Assumptions 1 and 2 can be enforced by the selection a large enough h and H, respectively. Particularly, $h \geq \max_{i\in N} |Q_i|$, and $H \geq \max_{i\in N} |S_i|$ ensures the completeness of our approach. However, in such a case, the complexity of the proposed approach meets the complexity of the straightforward solution in Section 3. A good guidance criterion for the choice of appropriate size of the receding horizon is the maximum, the average, or the mean of the shortest distance (i.e., the smallest number of transitions) between two actions labeled with non-silent service sets in the given TSs. If the selected horizon is too short, there is frequently no action labeled with a non-silent service set present in the intersection automaton, and the horizon gets frequently extended. If the horizon is slightly longer, the resulting intersection automata contain only a few actions labeled with non-silent service sets, and the backtracking might take place quite often. On the other hand, if the selected horizon is too long, then the intersection automaton might be too large to be efficiently handled. The goal is to select a horizon to achieve a reasonable size of the intersection automaton (according to our experience, hundreds to thousands of states maximally) while containing as many actions labeled with non-silent services as possible.

6. Example

To demonstrate our approach and its benefits, we consider the system from Example 1. We have implemented the proposed solution in MATLAB, and we illustrate snapshots of the resulting trace under stepwise synchronization in Fig. 2(A)–(D). It can be seen that the agents make progress towards satisfaction of their respective formulas. In the computation, the default values of planning horizons were $h = 3$, and $H = 5$. In several cases, the latter value had to be extended as described in Section 5. The maximum value needed in order to find a solution was $H = 9$. The sizes of the product automata handled in each iteration of the algorithm have significantly reduced in comparison to the straightforward centralized solution from Section 3, where all three agents belong to the dependency class yielding thus a synchronized TS with $144^3 \approx 3$ million states. With our dynamic decomposition, at most two agents belong to the same dependency class at the time, resulting into product automata sizes in order of thousands states and the computation of each iteration took seconds. When the agents are not dependent on each other within h the sizes of product automata are tens to hundreds states and the computation of each iteration took seconds to minutes. The durations of all agents' transitions were randomly generated from $\{5, \ldots, 10\}$ time units. The first 7 services in the plan of agent 2 have been completed after 54 iterations, with average duration of $\approx$477.1 time units. In the event-triggered solution, the individual resulting traces did not change, however, as indicated in Fig. 2(E), the randomized transition durations caused some of the agents progress more and some of them less in comparison to the stepwise solution. Average number of iterations to provide the 7 services of agent 2 was 30.3, and average time of completion was 494.2 time units (computed from 20 simulation cases). Finally, Fig. 2(F) shows the outcome of the event-triggered solution after 477 time units when the duration of transitions of agent 2 was changed to a random number in between $\{1, \ldots, 5\}$. The average number of iterations to provide the first 7 services of agent 2 was 26.3, and average time of completion was 355.6 time units (from 20 simulations). The outcome of the stepwise solution for this case did not change, except for the average time of completion, which is now $\approx$457.5. This case thus demonstrates better suitability of the event-triggered solution for heterogeneous multi-agent systems.

7. Summary and future work

We have proposed an automata-based receding horizon approach to solve the multi-agent planning problem from local

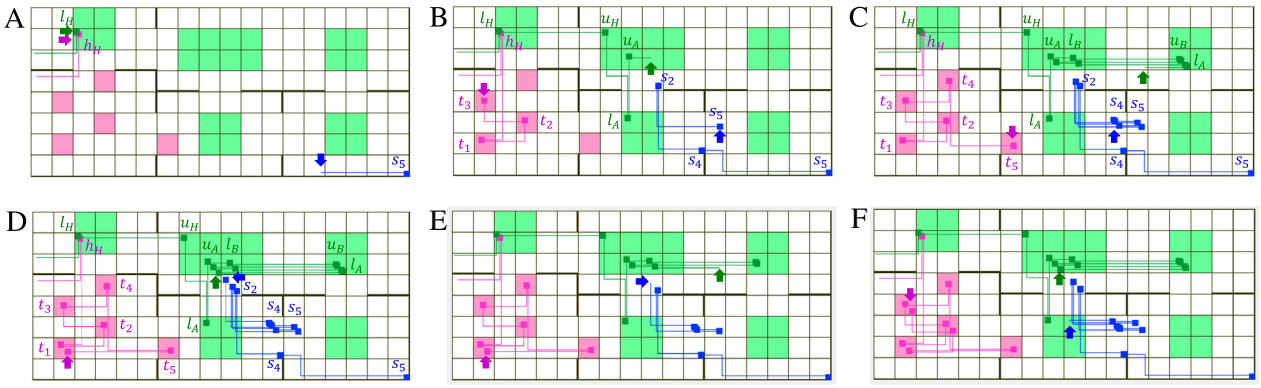

Fig. 2. Trace prefixes of agent 1 (green/light), agent 2 (pink/medium), and agent 3 (blue/dark). The initial position of the agents is in the bottom left corner of R_1, in the top left corner of R_3, and in the bottom right corner of R_5, respectively. The services are depicted as squares, the current position of the agents at the moment of the snapshot is indicated with arrows. In all figures, services l_H and h_H, and t_5 and s_4 are provided at the same time. Specifically, (A) depicts the moment when l_H and h_H are provided and (C) depicts the moment when t_5 and s_4 are provided. (A)–(D) the outcome of the stepwise solution after 1, 4, 6, and 7 services of agent 2 are provided, which is after 5, 22, 48, and 54 iterations, respectively; (E) an example outcome of the event-triggered solution after 7 services of agent 2, which is after 24 iterations; (F) an example outcome of the event-triggered solution after 477 time units in case agent 2 is faster in executing its transitions than the other two; 9 services were completed by agent 2 in 37 iterations. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

LTL specifications. The solution decomposes the infinite horizon planning problem into finite horizon planning problems that are solved iteratively. It enables each agent to restrict its focus only on the agents that are constrained by its formula within a limited horizon, and hence to decentralize the planning procedure. Moreover, via considering the finite horizon, we reduce the size of handled state space. Stepwise synchronization can be substituted with less frequent event-based synchronization, increasing the independence of the agents during the plan execution. Future research directions include involving various optimality requirements, or robustness to small perturbations as well as evaluation of the approach using mobile robots.

Acknowledgments

The authors are with the ACCESS Linnaeus Center, School of Electrical Engineering, KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden and with the KTH Centre for Autonomous Systems.

References

Baier, C., & Katoen, J.-P. (2008). *Principles of model checking*. MIT Press.
Belta, C., & Habets, L. C. G. J. M. (2006). Control of a class of nonlinear systems on rectangles. *IEEE Transactions on Automatic Control*, *51*(11), 1749–1759.
Bhatia, A., Maly, M. R., Kavraki, L. E., & Vardi, M. Y. (2011). Motion planning with complex goals. *IEEE Robotics & Automation Magazine*, *18*(3), 55–64.
Chen, Y., Ding, X. C., Stefanescu, A., & Belta, C. (2012). Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, *28*(1), 158–171.
Cormen, T. H., Stein, C., Rivest, R. L., & Leiserson, C. E. (2001). *Introduction to algorithms* (2nd ed). McGraw-Hill Higher Education.
Ding, X. C., Belta, C., & Cassandras, C. G. (2010). Receding horizon surveillance with temporal logic specifications. In *IEEE conference on decision and control* (pp. 256–261).
Filippidis, I., Dimarogonas, D. V., & Kyriakopoulos, K. J. (2012). Decentralized multi-agent control from local LTL specifications. In *IEEE conference on decision and control* (pp. 6235–6240).
Guo, M., & Dimarogonas, D. V. (2015). Multi-agent plan reconfiguration under local LTL specifications. *International Journal of Robotics Research*, *34*(2), 218–235.
Jing, G., Finucane, C., Raman, V., & Kress-Gazit, H. (2012). Correct high-level robot control from structured english. In *IEEE international conference on robotics and automation* (pp. 3543–3544).
Karaman, S., & Frazzoli, E. (2011). Vehicle routing with temporal logic specifications: Applications to multi-UAV mission planning. *International Journal of Robust and Nonlinear Control*, *21*, 1372–1395.
Kloetzer, M., & Belta, C. (2008). A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, *53*(1), 287–297.
Kloetzer, M., Ding, X. C., & Belta, C. (2011). Multi-robot deployment from LTL specifications with reduced communication. In *IEEE conference on decision and control and European control conference* (pp. 4867–4872).
Kress-Gazit, H., Fainekos, G. E., & Pappas, G. J. (2009). Temporal logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, *25*(6), 1370–1381.
LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.
Loizou, S. G., & Kyriakopoulos, K. J. (2005). Automated planning of motion tasks for multi-robot systems. In *IEEE conference on decision and control* (pp. 78–83).
Quottrup, M. M., Bak, T., & Zamanabadi, R. I. (2004). Multi-robot planning: a timed automata approach. In *IEEE International conference on robotics and automation* Vol 5. (pp. 4417–4422).
Svorenova, M., Tumova, J., Barnat, J., & Cerna, I. (2012). Attraction-based receding horizon path planning with temporal logic constraints. In *IEEE conference on decision and control* (pp. 6749–6754).
Tumova, J., & Dimarogonas, D. (2014). A receding horizon approach to multi-agent planning from local LTL specifications. In *American control conference* (pp. 1775–1780).
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. *International Journal of Robotics Research*, *32*(8), 889–911.
Wiltsche, C., Ramponi, F. A., & Lygeros, J. (2013). Synthesis of an asynchronous communication protocol for search and rescue robots. In *European control conference* (pp. 1256–1261).
Wongpiromsarn, T., Topcu, U., & Murray, R. M. (2010). Receding horizon control for temporal logic specifications. In *Hybrid systems: Computation and control* (pp. 101–110).

Jana Tumova received her Ph.D. degree in computer science from Masaryk University in Brno, Czech Republic in 2013. She is currently a postdoctoral researcher at the Automatic Control Department at KTH Royal Institute of Technology in Stockholm, Sweden. Her research interests include formal verification, control synthesis, and temporal logics in general, as well as formal methods applied in robot motion planning, control and analysis of dynamical and hybrid systems and multi-agent control.

Dimos V. Dimarogonas was born in Athens, Greece, in 1978. He received the Diploma in Electrical and Computer Engineering in 2001 and the Ph.D. in Mechanical Engineering in 2007, both from the National Technical University of Athens (NTUA), Greece. Between May 2007 and February 2009, he was a Postdoctoral Researcher at the Automatic Control Laboratory, School of Electrical Engineering, ACCESS Linnaeus Center, KTH Royal Institute of Technology, Stockholm, Sweden. Between February 2009 and March 2010, he was a Postdoctoral Associate at the Laboratory for Information and Decision Systems (LIDS) at the Massachusetts Institute of Technology (MIT), Boston, MA, USA. He is currently an Associate Professor at the Automatic Control Laboratory, ACCESS Linnaeus Center, KTH Royal Institute of Technology, Stockholm, Sweden. His current research interests include Multi-Agent Systems, Hybrid Systems and Control, Robot Navigation and Networked Control. He was awarded a Docent in Automatic Control from KTH in 2012. He serves in the Editorial Board of Automatica, the IEEE Transactions on Automation Science and Engineering and the IET Control Theory and Applications and is a member of IEEE and the Technical Chamber of Greece. He received an ERC Starting Grant from the European Commission for the proposal BUCOPHSYS in 2014 and was awarded a Wallenberg Academy Fellow grant in 2015.