Multi-Agent Planning under Local LTL Specifications and Event-Based Synchronization

Jana Tumova \(^{a}\), Dimos V. Dimarogonas \(^{a}\)

\(^{a}\) School of Electrical Engineering, KTH Royal Institute of Technology, SE-100 44 Stockholm

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.

Key words: Temporal logic, finite state machines, formal verification, path planning, synchronization, decentralized control, robot control.

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 \([3, 11, 13, 21]\): 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., \([11, 13, 14, 21]\) 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., \([2]\)). 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.

As a mission specification language, we use Linear Temporal Logic (LTL), for its resemblance to natural language \([9]\), 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 [18]. 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 [16], whereas in [12, 15], 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 [10], and a general reactivity LTL fragment has been used in [20]. Decentralized control of a robotic team from local LTL specification with communication constraints is proposed in [7]. 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 [4, 19], 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 [8], 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 [21] 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 [6, 17] for a single-agent LTL planning to achieve a locally optimal behavior.

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

2 Preliminaries

Given a set $S$, let $2^S$, and $S^*$ 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 $i$-th 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 $1 \leq j \leq i_j \leq i_{j+1}$. A factor of $w$ is a continuous, finite or infinite, subsequence $w(i)w(i+1) \ldots$, where $1 \leq i \leq l$. A prefix of $w$ is a finite factor starting at $w(1)$, and a suffix of $w$ is an infinite factor. $N$ and $\mathbb{R}_+^n$ denote positive integers and non-negative real numbers, respectively.

A transition system (TS) is a tuple $T = (S, s_{init}, A, T)$, where $S$ is a finite set of states; $s_{init} \in S$ is the initial state; $A$ is a finite set of actions; and $T \subseteq S \times A \rightarrow S$ is a partial deterministic transition function. For simplicity, we denote a transition $T(s, \alpha) = s'$ by $s \xrightarrow{\alpha} s'$. A trace of $T$ is an infinite alternating sequence of states and actions $\tau = s_1\alpha_1s_2\alpha_2 \ldots$, 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 $\tau$ that begins and ends with a state.

A linear temporal logic (LTL) formula $\phi$ over the set of atomic propositions $\Pi$ is defined inductively: (i) $\pi \in \Pi$ is a formula, and (ii) if $\phi_1$ and $\phi_2$ are formulas, then $\neg \phi_1$, $X \phi_1$, $\phi_1 \lor \phi_2$, $F \phi_1$, and $G \phi_1$ are each a formula, where $\neg$ and $\lor$ are standard Boolean connectives, and $X$, $U$, $F$, and $G$ are temporal operators. The semantics of LTL are defined over infinite words over $2^\Pi$. $\pi \in \Pi$ is satisfied on $w = \varpi_1 \varpi_2 \ldots$ if $\pi \in \varpi_1. X \phi$ holds true if $\phi$ is satisfied on the word that begins in the next position $\varpi_2$, $\phi_1 U \phi_2$ states that $\phi_1$ has to be true until $\phi_2$ becomes true, and $F \phi$ and $G \phi$ are true if $\phi$ holds on $w$ eventually, and always, respectively. We denote the satisfaction of $\phi$ on a word $w$ as $w \models \phi$. The set of all words accepted by an LTL formula $\phi$ is $L(\phi)$. For full details see, e.g., [1].

An automaton is a tuple $A = (Q, q_{init}, \Sigma, \delta, F)$, where $Q$ is a finite set of states; $q_{init} \in Q$ is the initial state; $\Sigma$ 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 $\delta^k(q)$ reachable from $q \in Q$ in exactly $k$ steps inductively as $\delta^0(q) = \{q\}$, and $\delta^k(q) =$
of T transitions. Each of the agents’ action executions takes a given by the transitions \( q \rightarrow q' \) of states and transition labels \( q_i \sigma_i q_{i+1} \ldots q_{k-1} \sigma_{k-1} q_k \), such that \( \forall i \leq j < k \), \((q_j, \sigma_j, q_{j+1}) \in \delta \). \( \text{dist}(q, q') \) denotes the length of the shortest path between \( q \) and \( q' \), i.e., the minimal number of states in a path \( q \rightarrow q' \). If no such path exists, then \( \text{dist}(q, q') = \infty \). If \( q = q' \), then \( \text{dist}(q, q') = 0 \). The shortest path can be computed using Dijkstra algorithm (see, e.g., [5]).

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 \( T_i = (S_i, S_{\text{inst}}, A_i, \mathcal{T}_i) \), 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 \( \mathcal{T}_i \) 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 \( \Delta_{s_i} \in \mathbb{R}^+ \) 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 \( \alpha \) 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 \( \text{sync}_i \) 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 \( \text{sync}_i \) 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 \( \text{sync}_{i'} \) 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 \( \text{nosync}_i \). The set of all synchronization requests of an agent \( i \) is thus \( \text{Sync}_i = \{\text{sync}_i, \text{nosync}_i\} \).

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

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 \( \Pi_i \). Services are provided within the agents’ transitions; each action \( \alpha \in A_i \) is labeled either with (i) a service set \( \varpi \in \Pi_i^\alpha \), provided upon the execution of \( \alpha \), or (ii) a special silent service set \( \mathcal{E}_i = \{\varepsilon_i\}, \varepsilon_i \notin \Pi_i \) indicating that \( \alpha \) 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 $\Pi_i$ and the action-labeling function $L_i : A_i \rightarrow 2^{\Pi_i} \cup 2^{E_i}$. Note that we specifically distinguish between a silent service and a non-silent service. The former transitions are labeled with $\delta_{s_i}$ of type $E_i$ and the latter transitions are. The former transitions are labeled with $\delta_{s_i}$ of type $E_i$ and the latter transitions are labeled with $\delta_{s_i}$. Non-silent services are associated with some of the latter transitions. The following hold: $\tau_i, t_{\alpha_i,j} = s_{i,1} \alpha_i 1, s_{i,2} \alpha_i 2, \ldots$, $\tau_i, t_{\alpha_i,j} = s_{i,1} \alpha_i 1, s_{i,2} \alpha_i 2, \ldots$, and $T_i = t_{s_{i,1} \alpha_i 1} t_{s_{i,2} \alpha_i 2} \cdots$.

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.

**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 $t_{\alpha_i,j} = \tau_i$. Then for all $i'$, there exists a matching index $j' \geq 1$, such that $r_{i',j'} = \tau_i$, and $t_{\alpha_i,j} = t_{\alpha_i,j}$. Furthermore, there exists $i' \in \mathcal{N}$, such that $t_{\alpha_i,j} = t_{\alpha_i,j}$, i.e., such that $\Delta_{\alpha_i,j} = 0$, for the matching index $j'$.

Suppose that the trace $\tau_i$, the synchronization sequence $\gamma_i$, and the transition time durations $\Delta_{\alpha_i,j}$ are all fixed for all $i \in \mathcal{N}$. Then there exists at most one collection of synchronization time durations $\Delta_{\alpha_i,j}$, $i \in \mathcal{N}$ that yields a set of compatible behaviors $\{\beta_i = (\tau_i, \gamma_i, T_i) | i \in \mathcal{N}\}$. 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 $\tau_i, \gamma_i$, and $\Delta_{\alpha_i,j}$, given for all $i \in \mathcal{N}$.

**Specifiﬁcations.** 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 $\phi_i$ over $\Pi_i = \bigcup_{i' \in D_i} \Pi_i$, 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 (every 4). 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 ($\phi_2 = (t_5 \land R_4 \land R_5 (\tau_2 \land X (t_5 \land R_4 \land R_5 \land R_5)))$). Agent 3 should patrol rooms $R_2, R_4, R_5$ ($\phi_3 = \bigwedge_{i \in \{2,4,5\}} \lnot \delta_i$).

Let us now introduce the notation needed for formalizing the specification satisfaction. Consider for a moment a single agent $M_i = (T_i, \Sigma_i, \Pi_i, L_i)$, and its behavior $\beta_i$, where, for simplicity of notation in the following paragraphs, we use $\beta_i = (\tau, \gamma, T_i)$, where $\tau = s_{1,1} \alpha_1 1 s_{2,2} \alpha_2 2, \ldots$, and $T_i = t_{s_{1,1} \alpha_1 1} t_{s_{2,2} \alpha_2 2} \cdots$. We denote by $r_i = \tau_i \alpha_i \gamma_i \tau_i \alpha_i \gamma_i \tau_i \alpha_i \gamma_i \ldots \in (2^{\Pi_i} \cup 2^{E_i})^\omega$ the unique service set sequence associated with $\tau_i$. The word
trace of
when
Definition 3 (Local LTL satisfaction) Let τ be a
trace of T_i with the word time sequence τ_i(t) = t_i1, t_i2, ...
The word produced by a set of compatible behaviors B_i = (β_i' | i' ∈ D_i) is w_B_i = ω_i1, ω_i2, ..., where ω_i = (∪j∈D_i v_j(t_j)) ∩ Π_i, for all j ≥ 1. The set of behaviors B_i is valid if w_B_i is infinite. The set of compatible behaviors B_i locally satisfies φ_i for the agent i, B_i |= φ_i, iff B_i is valid and w_B_i |= φ_i.

Note that even if B_i = B_i = B_j, it may be the case that w_B_i ≠ w_B_j and B_i |= φ, but B_j |= φ.

Problem 1 Consider a set of agents N = {1, ..., N}, and suppose that each agent i ∈ N is modeled as a tuple M_i = (T_i, Σ_i, Π_i, I_i), and assigned a task in the form of an LTL formula φ_i over Π_i = ∪i∈D_i Π_i', for some {i} ⊆ D_i ⊆ N. For each i ∈ N find a trace τ_i = s_{i,1}, α_{i,1}, s_{i,2}, α_{i,2}, ..., of T_i, and a synchronization sequence γ_i over Sync_i, with the property that regardless of the values of the transition time durations Δ_{s_{i,1}}, Δ_{s_{i,2}} ..., ∈ ℜ_+*, i ∈ N, the set of the agents’ behaviors {β_i' | i' ∈ D_i} locally satisfies φ_i, for all i ∈ N.

As each of the LTL formulas φ_i, i ∈ {1, ..., N} over Π_i can be algorithmically translated into a deadlock-free language equivalent BA [1], from now on, we pose the problem with the task specification of each agent i given as a BA B_i = (Q_i, q_{init,i}, δ_i, Σ_i, 2^{Π_i}, F_i) and the local task satisfaction condition formulated as w_B_i ∈ L(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 Prob. 1 can be obtained by imposing full synchronization and modifying the standard control plan synthesis procedure for TS from LTL specification (see, e.g., [11]): (1) The set of agents is first partitioned into dependency classes similarly as in [8] by the iterative application of the rule that if i' ∈ D_i, then i' belongs to the same dependency class I_i as i; (2) For each dependency class I_i = {i_1, ..., i_k}, and each agent i ∈ I_i, we set γ_i = r_{i_1}r_{i_2}..., where r_{i,j} = sync_i. This step yields compatible behaviors of all agents in N independently of their traces and transition time durations. (3) For each I_i = {i_1, ..., i_k}, a TS T_i with S_i = S_{i_1} x ... x S_{i_k} is constructed that represents the stepwise-synchronized traces of the agents within the class. (4) A product P_i of T_i and B_{i_1}, ..., B_{i_k} is built that captures only the agents’ traces that are admissible by T_i and result into behaviors that locally satisfy φ_1, ..., φ_k, respectively. The product P_i is analyzed using graph algorithms to find a solution that projects onto the desired traces of T_i, for all i ∈ I_i. The outlined procedure is correct and complete. Although a certain level of decentralization is achieved via dependency partition, the algorithm suffers from an 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 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 Prob. 1. In Sec. 4.1, we set the preliminary synchronization sequences to be followed. In Sec. 4.2 we present a finite horizon planning 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. Sec. 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 Sec. 3.

### 4.1 Preliminary Synchronization Sequence

We set $\gamma_i = r_{i,1} r_{i,2} \ldots$, where $r_{i,j} = \text{sync}_i$, 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 Def. 1 and Def. 2:

**Lemma 1.** Let $\gamma_i = r_{i,1} r_{i,2} \ldots$, where $r_{i,j} = \text{sync}_i$, for all $i \in \mathcal{N}$. For traces $\tau_i = s_{i,1} \alpha_1,1 \ldots \tau_i = s_{i,1} \alpha_{i,t},1 \ldots$ of $\mathcal{T}_i, \mathcal{T}_i, i \in \mathcal{N}$, 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 $\mathcal{M}_i = (\mathcal{T}_i, \text{Sync}_i, \Pi_i, I_i)$ and the specification BAs $\mathcal{B}_i = (Q_i, q_{\text{init},i}, \delta_i, \Sigma_i = 2^\Pi_i, F_i)$, for all $i \in \mathcal{N}$, the inputs to the finite horizon planning algorithm are: the current states of $\mathcal{T}_i, \ldots, \mathcal{T}_N$, denoted $s_1, \ldots, s_N$, initially equal to $s_{\text{init},1}, \ldots, s_{\text{init},N}$, respectively; the current states of $\mathcal{B}_1, \ldots, \mathcal{B}_N$, denoted $q_1, \ldots, q_N$, initially equal to $q_{\text{init},1}, \ldots, q_{\text{init},N}$, 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 $\Phi = \{\mathcal{B}_1, \ldots, \mathcal{B}_N\}$ into the smallest possible subsets $\Phi_1, \ldots, \Phi_M$, such that none of the transitions of any $\mathcal{B}_i \in \Phi_i$ that appears within the horizon $h$ from the current state $q_i$ of $\mathcal{B}_i$ imposes restrictions on any agent $\mathcal{B}_j$, where $\mathcal{B}_j \notin \Phi_i$. This partition corresponds to the necessary and sufficient dependency up to the horizon $h$.

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

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

$$\Pi_i^h(q) = \bigcup_{q' \in \delta_i(q), 0 \leq j \leq h} \{\Pi_{i'} \mid \Pi_{i'} \text{ participating in } q'\},$$

where $\delta_i(q)$ denotes the set of states reachable from $q$ in $j$ steps (see Sec. 2).

**Definition 5 (Dependency partition)** Given that $q_1, \ldots, q_N$ are the respective current states of BAs $\mathcal{B}_1, \ldots, \mathcal{B}_N$, the dependency partition of $\Phi$ is induced by the equivalence $\sim^h$ defined on $\Phi$: (i) if $\mathcal{B}_i \sim^h \mathcal{B}_i$, and (ii) if there exists $\mathcal{B}_i$, such that $\mathcal{B}_i \sim^h \mathcal{B}_j$, and $\Pi_{i'} \subseteq \Pi_{i'}^h(q_{i'})$ or $\Pi_{i'} \subseteq \Pi_{i'}^h(q_{i'})$, then also $\mathcal{B}_i \sim^h \mathcal{B}_i$.

The dependency partition of $\Phi$ is then $\{\Phi_1, \ldots, \Phi_M\}$, where $(\mathcal{B}_i \sim^h \mathcal{B}_i') \iff (\mathcal{B}_i \notin \Phi_{i'} \iff \mathcal{B}_{i'} \in \Phi_{i'})$.

The dependency partition of the set of all agents $\mathcal{N}$ is $\mathcal{I} = \{I_1, \ldots, I_M\}$, where a dependency class $I_\ell$ 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 $q_1, \ldots, q_N$. Hence it is dynamically recomputed in each iteration of our solution. Although the local satisfaction of $\phi_i$ for $i \in I_i$ depends on the traces of all the agents $j \in D_i$, it is not influenced by the agents $j \in D_i \setminus I_i$ within the horizon $h$. Thus, when we now concentrate on planning within the horizon $h$, we can safely treat each dependency class $I_\ell = \{i_1, \ldots, i_\ell\}$ separately. In the worst case, the dynamic dependency partition equals to the offline one from Sec. 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 Sec. 6.

**Intersection Automaton.** For each dependency class $I_\ell$, we construct a finite automaton that represents the language intersection of the BAs in $\Phi_\ell = \{\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_\ell}\}$ up to the pre-defined horizon $h$. In this step, we rely on the fact that the preliminary synchronization sequence set in Sec. 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 $\Phi_\ell$ be ordered according to $\prec$, i.e. $i_\ell \prec i_j$, for all $1 \leq i < j \leq n$.

**Definition 6 (Intersection automaton)** The intersection automaton of $\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_\ell}$ up to the horizon $h$ is $\mathcal{A}_h = (Q_A, q_{\text{init},A}, \Sigma_A, \delta_A, F_A)$, where $Q_A \subseteq Q_{i_1} \times \ldots \times Q_{i_\ell} \times \mathbb{N}$ is a finite set of states, generated as described below; $q_{\text{init},A} = (q_{i_1}, \ldots, q_{i_\ell}, 1)$; $\Sigma_A = \{\cup_{i \in I_i} \sigma_{ii} \mid \sigma_{ii} \in (2^{\Pi_i} \cup 2^c)\}$; Let $Q_0 = \{q_{\text{init},A}\}$.
For all $1 \leq j \leq h$, we define $(q_{1j}, \ldots, q_{nj}, k') \in Q_A^j$ and $((q_{1j}, \ldots, q_{nj}, k), (q_{1j}', \ldots, q_{nj}', k')) \in \delta_A$ iff (i) $(q_{1j}, \ldots, q_{nj}, k) \in Q_A^{-1}$, (ii) for all $i \in I$, either $(q_i, \sigma_i \cap \Pi_i, q_i') \in \delta_i$ or $q_{i} = q_{i}'$, and $\epsilon_i \in \sigma$, and (iii) $k' = k + 1$ if $q_{ij} \in F_{\epsilon_i}$, where $k = k \mod n_\Pi$ and $k' = k$ otherwise. Finally, $Q_A = \bigcup_{0 \leq j \leq h} Q_A^j$ and $\delta_A = \bigcup_{1 \leq j \leq h} \delta_A^j$.

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 Sec. 2). A path in $A^h$ from the initial state $(q_{1j}, \ldots, q_{nj}, 1)$ to a state $(q_{1j}, \ldots, q_{nj}, k)$ corresponds to a path from $q_i$ to $q_j$ in each $B_{i'}$ and vice versa. Formally, these two lemmas from the construction:

**Lemma 2** Consider a path $\rho_i = q_{i1}, q_{i2}, \ldots, q_{in_i}, k \in A^h$, where $q_i$ denotes the tuple $(q_{1j}, \ldots, q_{nj}, k, j) \in Q_A$, for all $1 \leq j \leq m$. Let $\omega_i = \omega_{i1}, \omega_{i2}, \ldots, \omega_{in_i}$ be the sequence of non-silent elements of $\omega_{i1}, \omega_{i2}, \ldots, \omega_{in_i}$. Finally, let $\omega_{i1}, \omega_{i2}, \ldots, \omega_{in_i}$ be the sequence of non-silent elements of $\omega_{i1}, \omega_{i2}, \ldots, \omega_{in_i}$. Then there exists a corresponding run $\rho_i = q_{i1}, q_{i2}, \ldots, q_{in_i}, k \in A^h$, where $q_i = q_{i1}$, $q_{i2}$, $\ldots$, $q_{in_i}$, $k$, such that $q_i \neq q_{ij}$ for any $1 \leq j \leq m$.

**Lemma 3** Consider a run $\rho_i = q_{i1}, q_{i2}, \ldots, q_{in_i}, k \in A^h$ over a word $w_i = \omega_{i1}, \omega_{i2}, \ldots, (2\Pi_i)^n$, where $q_{i1} = q_{i1}$. Let $\sigma_1, \ldots, \sigma_{n_i}$ be a word over $\Sigma_A$ with the property that there exists its subsequence $\sigma_1, \ldots, \sigma_j$, such that $\sigma_1 \cap \Pi_i = \omega_{i1}, \omega_{i2}, \ldots, \omega_{in_i}, k \in A^h$, for all $1 \leq j \leq m$, and for $j \geq 1$, $\epsilon_i \neq j$, for any $j \geq 1$. Then there exists a path $q_i, q_{i2}, \ldots, q_{in_i}, k \in A^h$, where $q_i$ denotes the tuple $(q_{1j}, \ldots, q_{nj}, k, j) \in Q_A$, such that: (i) $q_{i1} = q_{i1}$, (ii) $q_{i2}, q_{i3}, \ldots, q_{in_i}, k \in A^h$, for all $1 \leq j \leq h$, and (iii) $q_{i2}, q_{i3}, \ldots, q_{in_i}, k \in A^h$, for all $1 \leq j \leq h$, where $q_{ij} \neq q_{ij}'$.

Through $k$, we remember the progress towards the satisfaction of the individual specifications ordered according to $\prec$; for $k \leq n_\Pi$, an accepting state is guaranteed to be present on a projected finite path of each $B_{i'}$, $1 \leq i \leq k$. For $k > n_\Pi$, an accepting state is surely present on each projected run of each $B_{i'}$, at least $\lceil \frac{k}{n_\Pi} \rceil$-times for all $1 \leq i \leq k$, and at least $\lceil \frac{k}{n_\Pi} \rceil$-times for all $k < n_\Pi$, where $k = k \mod n_\Pi$. To be able to identify “profitable” actions/translations of the agents w.r.t. $\Phi$, and thus also w.r.t. $\Phi$, we assume that at least a state which represents a step towards the satisfaction of the highest-order specification $B_{i'}$ is present in $A^h$. In Sec. 4.2 this fact will allow us to set short-term goals in TSS $T_{ij}$, $T_{ij}$. We discuss its relaxation in Sec. 5.

**Assumption 1** Assume that $F_A$ is not empty.

**Definition 7 (Automaton progressive function)** The progressive function $V_A : Q_A \to \mathbb{N}_0 \times \mathbb{Z}_0^*$ is for a state $q = (q_{1j}, \ldots, q_{nj}, k)$ defined as: $V_A(q) = (k, -\min_{q_j \in F_A} \dist(q, q_j))$.

The increasing value of $V_A$ indicates a progress towards the satisfaction of the individual local specifications in $\Phi_i$, ordered according to $\prec$. No progress can be achieved from a state $q$, such that $V_A(q) = (k, -\infty)$ within the horizon $h$, and hence, we remove such states from $A^h$. From Assump. 1, $V_A(q_{\text{init}}, A) = (1, d)$, 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 $I_j$ 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 $H$ and $A^h$ from Def. 6 is an automaton $P_h = (Q_P, q_{\text{init}}, p, \Sigma_P, \delta_P)$, where $Q_P \subseteq S_{i1} \times \ldots \times S_{iH} \times Q_A$ is a finite set of states, generated as described below; $q_{\text{init}} = (q_{1j}, \ldots, q_{nj}, q_{\text{init}}A)$; $\Sigma_P = \{A_{i1} \times \ldots \times A_{iH} \times \Sigma_A\};$ Let $Q_P = \{q_{\text{init}}\}$. For all $1 \leq j \leq H$, $q = (q_{1j}, \ldots, q_{nj}, q_{1j}, \ldots, q_{nj}, \sigma_j) = (q_{1j}, \ldots, q_{nj}, q_{1j}, \ldots, q_{nj}, \sigma_j) \in \Sigma_P$. Define that $q' \in Q_P$ and $(q, \sigma, q') \in \delta_P$ iff (i) $q \in Q_P$, (ii) for all $i \in I$, it holds that $T(s_i, \omega_i) = s_{ij}$, and (iii) $(q_{\text{init}}, q_{1j}, q_{1j}) \in \delta_A$. Finally, $Q_P = \bigcup_{0 \leq j \leq H} Q_P^j$ and $\delta_P = \bigcup_{1 \leq j \leq H} \delta_P^j$.

The set of accepting states $F_P$ is not significant for the further computations, hence we omit it from $P_h$. Analogously as $A^h$, $P_h$ can be viewed as a graph (see Sec. 2). A path in $P_h$ can be projected onto a finite trace prefix of each $T_{ij}$, a finite run prefix of $A^h$ and further through Lemma 2 onto a finite run prefix of each BA $B_{i'}$, too.

**Definition 9 (Projection)** Consider a path $q = q_{1j}, q_{2j}, \ldots, q_{nj}, \ldots, q_{m-1j}, q_{mj}, q_{mj}$ in $P_h$, where $q_i = q_{\text{init}}$, $p$ can be projected onto a finite trace prefix $\tau_i(q)$ of each $T_{ij}$, $i \in I$ in the expected way: for all $1 \leq j \leq m$, the $j$-th state and action of $\tau_i(q)$ is $q_{ij}$, and $\alpha_{ij}$ if the $j$-th state and transition label of $q$ is $q_j = (s_{nj}, \ldots, s_{nj}, q_{A,i})$. 

7
and $\sigma_j = (\alpha_1, \ldots, \alpha_{n_j}, \sigma_{A,j})$, respectively. Furthermore, for all $1 \leq j \leq m$, the $j$-th state of the projected run prefix $\hat{\rho}_i(g)$ of $A^h$ is $q_{A,j}$ if the $j$-th state of $g$ is $q_i = (s_1, \ldots, s_{n_i}, q_{A,j})$; and assuming that $q_{A,j} = (q_{i,j}, \ldots, q_{n_i,j}, k)$, the state of the corresponding state sequence $\hat{\rho}_i(g)$ in $B_i$ is $q_{i,j}$.

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

**Definition 10 (Progressive function and state)**

The progressive function $V_P : Q_P \rightarrow \mathbb{N}_0 \times \mathbb{Z}_0^*$ is inherited from the interaction automaton $A^h$ (Def. 7), i.e., for all $q = (s_1, \ldots, s_{n_i}, q_A) \in Q_P$, $V_P((s_1, \ldots, s_{n_i}, q_A)) = V_A(q_A)$. A state $q \in Q_P$ is a progressive state if $V_P(q) > V_P(q_{init,p})$. A maximally progressive state is a progressive state $q$, with the property that for all $q' \in Q_P$, it holds $V_P(q) \geq V_P(q')$.

**Finite Horizon Plan Synthesis.** To find a suitable finite horizon plan, we impose the following assumption and discuss its relaxation in Sec. 5. 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 $\phi_{i_t}$.

**Assumption 2** Assume that there is a progressive state $q_p$ reachable in $P^H$ via a finite path $q_1 \sigma_1 \ldots \sigma_m q_m$, where $q_1 = q_{init,p}$, $q_m = q_p$, and $L(\alpha_{1,j}) \neq 0$, for some $\sigma_j = (\alpha_{i_t,j}, \ldots, \alpha_{n_t,j}, \sigma_{A,j})$, $1 \leq j \leq m - 1$.

We compute a suitable finite horizon plan as the shortest path $\hat{\rho} = q_1 \sigma_1 q_2 \ldots q_m \sigma_{m-1} q_m$ in $P^H$ from $q_1 = q_{init,p}$ to a maximally progressive state $q_m = q_{max}$. Such a path can be found in linear time with respect to the size of $P^H$ (see Sec. 2). The projection of $\hat{\rho}$ onto the individual TSSs gives finite trace fragments $\hat{\tau}_i(\hat{\rho}) = s_{i,1} \sigma_{i,1} s_{i,2} \ldots s_{i,m-1} \sigma_{i,m-1} s_{i,m}$, to be followed by each agent $i_t \in I_t$. Alg. 1 summarizes the proposed finite horizon planning for the set of all agents $N$. Since this algorithm will be run iteratively, starting from its second execution on, it also takes as an input the fragments $\tau_1, \ldots, \tau_N, \hat{\rho}_1, \ldots, \hat{\rho}_N$ and the progressive function value of the maximally progressive state from the previous iteration. In comparison to the solution from Sec. 3, we treat separately each dynamic dependency class (lines 2–13). These classes are only 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 Sec. 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 $i \in N$ first sets its preliminary synchronization sequence $\gamma_i = r_{i,1} r_{i,2} \ldots$ and synchronizes with the other agents, i.e., sends $r_{i,1} = syn_{i,1}$, and waits for the reception of $syn_{i',j}$ from all $i' \in N$. Second, it computes the finite horizon plans by Alg. 1. Third, it executes the first action $\alpha_{i,1}$ of the planned trace fragment $\tau_i = s_{i,1} \sigma_{i,1} s_{i,2} \ldots s_{i,m-1} \sigma_{i,m-1} s_{i,m}$ 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 $s_i$ is updated. The previous synchronization through $r_{i,1}$ has ensured that each agent $i' \in I_t$ executes its respective action $\alpha_{i',1}$ simultaneously. At the same time, the current state of the BAs $B_i$ is updated to the second state $q_{i,2}$ of the run fragment $\hat{\rho}_i$. Note that if $L(\alpha_{i,1}) = \varepsilon$, then $q_{i,2} = q_{i,1}$ by Def. 6 and Def. 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 $B_i$ has been made and in the next iteration, progressing towards another agent’s specification will be prioritzed. 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
Algorithm 2 Stepwise solution to Prob. 1

Input: a set of agents \( \mathcal{N} = \{1, \ldots, N\} \), their models \( M_1, \ldots, M_N \); BAs \( B_1, \ldots, B_N \); 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 \( B_1, \ldots, B_N \).

1: for all \( i \in \mathcal{N} \) do
2: \( \tau_i := (1, \ldots, N) \); \( s_i := s_{\text{init}, i} \); \( q_i := q_{\text{out}, i} \);
3: \( \tau_i := \text{empty}; \rho_i := \text{empty}; V_{\rho_{\max}} := (0, 0) \);
4: \( \tau_i := \text{empty; \text{sync}} \); wait for sync from all \( i' \in \mathcal{N} \);
5: \( j_i := 2 \);
6: end for

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 in the planned trace fragment \( \tau_i \) of an agent \( i \) are all associated with silent services, then this agent does not need to synchronize with the others nor to recomputation 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 \( \tau_i = s_{i,1} \alpha_{i,1}s_{i,2} \alpha_{i,2} \ldots s_{i,m} \) and \( \rho_i = q_{i,1}q_{i,2} \ldots q_{i,m} \), 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 \( t_{s_{i,m}} \) right before the execution of the action \( \alpha_{i,j} \) if \( \gamma_{i,j} > 1 \) with one of the following properties: \( \alpha_{i,1} \) is non-silent, or \( q_{i,1} \) is accepting, or \( q_{i,j} = q_{i,m} \), or there exists an agent \( i' \), such that \( \text{sync}_{i'} \) has been received during \( (t_{s_{i,j-1}}, t_{s_{i,j}}) \). If one of the conditions is met, the agent sends \( \text{sync}_{i} \), and waits for receiving \( \text{sync}_{i'} \), 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 \( \text{sync}_{i} \) planned within the preliminary synchronization sequence \( \gamma_i \) with \( \text{nosync}_{i} \). Note that thanks to the enforced compatibility of the behaviors, a deadlock is prevented. The solution is summarized in Alg. 3.

Algorithm 3 Event-triggered solution to Prob. 1

Input: a set of agents \( \mathcal{N} = \{1, \ldots, N\} \), their models \( M_1, \ldots, M_N \); BAs \( B_1, \ldots, B_N \); 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 \( B_1, \ldots, B_N \).

1: for all \( i \in \mathcal{N} \) do
2: \( \tau_i := (1, \ldots, N) \); \( s_i := s_{\text{init}, i} \); \( q_i := q_{\text{out}, i} \);
3: \( \tau_i := \text{empty; \text{sync}} \); wait for sync from all \( i' \in \mathcal{N} \);
4: \( j_i := 2 \);
5: end for

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 \( \tau_i = s_{i,1} \alpha_{i,1}s_{i,2} \alpha_{i,2} \ldots s_{i,m} \) and \( \rho_i = q_{i,1}q_{i,2} \ldots q_{i,m} \), 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 \( t_{s_{i,m}} \) right before the execution of the action \( \alpha_{i,j} \) if \( \gamma_{i,j} > 1 \) with one of the following properties: \( \alpha_{i,1} \) is non-silent, or \( q_{i,1} \) is accepting, or \( q_{i,j} = q_{i,m} \), or there exists an agent \( i' \), such that \( \text{sync}_{i'} \) has been received during \( (t_{s_{i,j-1}}, t_{s_{i,j}}) \). If one of the conditions is met, the agent sends \( \text{sync}_{i} \), and waits for receiving \( \text{sync}_{i'} \), 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 \( \text{sync}_{i} \) planned within the preliminary synchronization sequence \( \gamma_i \) with \( \text{nosync}_{i} \). 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 Sec. 3. Then for each agent \( i \in I_k \), we can replace \( \text{sync}_{i} \) with \( \text{sync}_{i}(I_k) \), and every \( \text{nosync}_{i} \) with \( \text{sync}_{i}(\{i\}) \). 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 $\rho_i$ contains infinitely many states $f_i \in F_i$ for all $i \in N$.

Proof. Denote $\tau_i = s_{i,1} \tau_{i,2} \ldots \rho_i = q_i \tau_{i,2} \ldots$ and $T(\tau_i) = t_{i,1} t_{i,2} \ldots$ for all $i \in N$. Consider a time instance $t_{i,j}$, where $j \geq 1$ and assume that $i$ is the most prioritized agent at $t_{i,j}$, i.e., that $i < i'$, for all $i' \in N$. Let $q$ be the path to a maximally progressive state $q_{\text{max}}$ of $P^H$ at time $t_{i,j}$ computed on line 4 of Alg. 1 and let $\hat{\tau}_i$ and $\hat{\rho}_i$ be the corresponding finite trace prefix of $\tau_i$ and the state sequence of $B_i$ computed on lines 7-9. Assume for a moment that the dependency partition $D$ remains the same at time $t_{i,j+1}$. Regardless of the steps taken in Alg. 2, the state $q_{\text{max}}$ is also present in $P^H$ at time $t_{i,j+1}$, hence if condition on line 5 is not satisfied, this state remains the progressive goal. Now, let $q'$ be the path to a maximally progressive state $q_{\text{max}}'$ of $P^H$ at time $t_{i,j+1}$ and let $\tilde{\tau}_i$ and $\tilde{\rho}_i$ be the corresponding finite trace prefix of $\tau_i$ and the state sequence of $B_i$. If $q_{\text{max}}' \neq q_{\text{max}}$, then $\mathbb{V}_p(q_{\text{max}}') > \mathbb{V}_p(q_{\text{max}})$ and $q_{\text{max}}'$ is, loosely speaking, closer to reaching an accepting state of $B_i$. Thanks to Assump. 1 and Assump. 2 and the finite number of states of $P^H$, $\mathbb{V}_p(q_{\text{max}}')$ is, inductively, a state of $P^H$ that projects onto an accepting state of $B_i$. The remainder of the proof is analogous to that of Lemma 5.

Lemma 5 The traces $\tau_1, \ldots, \tau_N$, where $\tau_i = s_{i,1} \alpha_{i,1} \ldots$ for all $i \in N$, 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 $\Delta_{a_{i,1}}, \Delta_{a_{i,2}} \ldots$

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 $\rho_i$ contains infinitely many states $f_i \in F_i$ for all $i \in N$.

Proof. Denote $\tau_i = s_{i,1} \alpha_{i,1} \ldots \rho_i = q_i \tau_{i,2} \ldots$ and $T_i = t_{i,1} t_{i,2} \ldots$ for all $i \in N$. To prove the lemma, we prove that at each time $t$, it holds that an accepting state $f_i \in F_i$ will eventually be reached for the most prioritized agent $i$ at time $t$. First, consider $t = 0$. Without loss of generality, assume that $i < i'$, for all $i' \in N$. Denote $I_i \subseteq I$ the dependency class $i$ belongs to. Let $\tau_i, \gamma_i$ and $\rho_i$ be the finite trace prefixes of $\tau_i, \gamma_i$, and the state sequences of $B_i, \ldots, B_N$ computed on line 9 of Alg. 3, respectively. Specifically for $I_i$, these were obtained from the projections of the shortest path $q_i$ that leads to a maximally progressive state $q_{\text{max},i} \ell = (s_{i,1}, m, \ldots, s_{i,m}, (q_{i,1}, m, \ldots, q_{i,m}, k_m))$ of $P^H$ computed on line 4 of Alg. 1. Let the execution proceed as described in Alg. 3 and let $f_i$ 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 $f_i$ is finite and after the requested synchronization is performed, the time $t_i$ equals to some $t_i'$ in $\bar{T}_i$, for all $i' \in N$. Let $\bar{q}_i$ and $\bar{q}_i'$ denote the respective states of $T_i$, and $B_i$ at time $t_i$, and let $\bar{\tau}_i$ and $\bar{\rho}_i$ denote the respective suffixes of $\tau_i$ and $\rho_i$ starting at $s_{i,m}$ and $q_{i,m}$, for all $i' \in N$. Denote by $I_{i'}$ the dependency class $i'$ belongs to at time $t_i$, and by $P^H_{i'}$ the corresponding product system. We now show the existence of a mapping between the suffixes $\bar{\tau}_i$ and $\bar{\rho}_i$ of agents $i' \in I_{i'} \cap I_i$ onto a single finite path $q_{i'}$ in $P^H_{i'}$, whose length is strictly smaller than the length of $q_i$ in $P^H$. Trivially, if $s_{i,m} = s_{i',m}$, for all $i' \in I_{i'} \cap I_i$, the path $q_{i'}$ is empty. Suppose that $s_{i,m} \neq s_{i',m}$, for some $i' \in I_{i'} \cap I_i$. We propose $q_{i'}$ to be the path whose projections onto the states of $T_i$ and $B_i$ are the respective sequences $s_{i'} \ldots s_{i,m} q_{i'}$ and $q_{i'} \ldots q_{i,m} \bar{\rho}_i$, such that the length of these two sequences, and hence also the length of $q_{i'}$, equals to the length of the suffix $\bar{\tau}_i$ that is the longest one among the agents of $i' \in I_{i'} \cap I_i$. This path is strictly shorter than $q_i$. At the same time, it exists in $P^H_{i'}$ due to the assumption that $s_{i,m} \geq s_{i',m}$, for all $s_{i'} \in S_{i'}$, $i' \in N$, and some $\alpha \in A_{i'}$. The existence of $q_{i'}$ ensures that a progress towards some $f_i \in F_i$, Thanks to Assump. 1 and Assump. 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 $q_{i'}$ will be the maximally progressive state $q_{\text{max},i'}$ in $P^H_{i'}$, until this state is reached. Inductively, a state $q_{i'}$ that projects onto an accepting state of $B_{i'}$ will eventually
ally be reached. At the same time, at least one non-silent service is provided by \( T_i \) on the path to \( q_i \). Furthermore, lines 18–21 of Alg. 3 ensure, that each \( i \in N \) will repeatedly become the most prioritized one. Altogether, we \( P_i \) contains infinitely many states \( f_i \), for all \( i \in N \).

**Theorem 2** The traces \( t_1, \ldots, t_N \) together with the synchronization sequences \( \gamma_1, \ldots, \gamma_N \) returned by Alg. 2 provide a solution to Prob. 1.

The complexity of one iteration of the solution is linear with respect to the size of \( P^H \), as the applied graph search algorithms are linear (see, e.g. [5]). The size of the intersection automaton \( A^h \) is in \( O(n_{l^h+1}) \), where \( n \) is the maximal set of states reachable in some \( B_i, i \in I_H \) within horizon \( h \). The size of the product \( P^H \) is \( O(n_{l^h+1}) \), where \( n \) is the maximal set of states reachable in some \( B_i \) or \( T_i, i \in I_H \) within the horizon \( H \). In the worst case, when \( n \) reaches the sizes of \( B_i \) or \( T_i \) respectively, and when the number of dependency classes \( n_T = 1 \), the complexity of one iteration reaches the one of the straightforward solution, i.e., the complexity of one iteration is in \( O(N \prod_{i \in N} |T_i| \cdot |B_i|) \), where \( |T_i| \) and \( |B_i| \) is the size of \( T_i \) and \( B_i \) respectively. However, as we demonstrate in Sec. 6, a dramatic improvement of computational times can be achieved in practice.

Assump. 1 may be violated for two different reasons: First, if the selected horizon \( h \) is too short, and although \( F_A = \emptyset \) in \( A^h \), there exists \( h' > h \), such that \( F_A \neq \emptyset \) in \( A^{h'} \). Second, if \( F_A = \emptyset \) even for \( h \to \infty \). We propose to systematically extend the horizon \( h \) and update the automaton \( A^h \) until a set of states \( F_A \) becomes nonempty, or until the extension does not change the automaton \( A^h \) any more. In the former case, the automaton \( A^h \) with the extended horizon satisfies Assump. 1 and thus is used in constructing \( P^H \), maintaining the remainder of the solution as described in Sec. 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 Assump. 1 holds, there is only one reason for violation of Assump. 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 \( T_1, \ldots, T_N; \) BAS \( B_1, \ldots, B_N \),

**System execution prefix** \( (t_1^i, \ldots, t_N^i, \rho_1, \ldots, \rho_N) \) up to the current time \( t \), where \( t_1^i = s_{i,1}, \ldots, t_N^i, \rho_1, \ldots, \rho_N \), for all \( i \in \{1, \ldots, N\} \).

**Output:** Updates to BAS \( B_1, \ldots, B_N \)

1. \( k := t \)
2. while solution not found do
3. \( k := k - 1 \)
4. Check, if the execution of \( \bigcup_{i \in \{1, \ldots, N\}} \omega_{t,k} \) can lead to a different set of states of BASs than to \( q_{i,t}, \ldots, q_{N,t} \).
5. If so, apply the change and go to line 6.
6. Execute one iteration of Alg. 1 from \( s_1 = s_1, \ldots, s_N = s_N, q_1 = q_1, \ldots, q_N = q_N \)
7. if a plan was found in line 5, continue with execution of Alg. 1, otherwise go to line 2 of Backtracking.
8. end while

Note, that Assump. 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 Sec 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 by the algorithm from Sec. 4.5.1. If the horizon is slightly longer, the resulting intersection automata contain only a few actions labeled with non-silent service sets, and the backtracking from Sec. 4.5.2 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 Sec.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 Sec. 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 between in $\{1, \ldots, 5\}$. 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 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.

Acknowledgements

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. This work was supported by the EU STREP RECONFIG, and by the H2020 ERC Starting Grant BUCOPHSYS.

References

Fig. 2. Trace prefixes of agent 1 (green/light), agent 2 (pink/medium), and agent 3 (blue/dark). The initial position of the agents are 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 are 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) depict 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.


