A Receding Horizon Approach to Multi-Agent Planning from Local LTL Specifications

Jana Tůmová and Dimos V. Dimarogonas

***Abstract*— We study the problem of control 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. Particularly, we consider that the task specification takes a form of a linear temporal logic formula, which may contain requirements and constraints on the other agent's behavior. A traditional automata-based approach to multi-agent strategy synthesis from such specifications builds on centralized planning for the whole team and thus suffers from extreme computational demands. In this work, we aim at reducing the computational complexity by decomposing the strategy synthesis problem into short horizon planning problems that are solved iteratively, upon the run of the agents. 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.**

I. Introduction

In recent years, a considerable amount of attention has been devoted to automatic synthesis of robot controllers to execute complex, high-level mission, 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 this problem and its variants rely on a three-step hierarchical procedure [3], [12], [14], [21]: First, the dynamics of the robotic system is abstracted into a finite, discrete transition system using e.g., sampling or cell decomposition methods based on triangulations or rectangular partitions. Second, leveraging ideas from formal verification methods, a discrete plan that meets the mission is synthesized. Third, the discrete plan is translated into a controller for the original system.

In this work, we focus on a multi-agent version of the above problem. Namely, we consider a team of robots, that are assigned a temporal mission each. As the robots may not be able to accomplish the mission without the help of the others, the individual mission specifications may contain requirements or constraints on the other team members' behavior. For instance, consider a warehouse solution with two mobile robots that periodically load and unload goods in certain locations of the warehouse. A part of the first robot's mission is to load an object in region A, however it is not able to load it by itself. Therefore at that point, the part of the mission is also a task for the second robot, to help loading.

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. tumova, dimos@kth.se. This work was supported by the EU STREP RECONFIG.

The goal of this paper is to synthesize a plan for each agent, such that each agent's mission specification 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, methods for discrete modeling of robotic systems can be used (e.g., [12], [14], [15], [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 (e.g., [2]). As a mission specification language, we use Linear Temporal Logic (LTL), for its resemblance to natural language [10], and expressive power.

Multi-agent planning from temporal logic specification has been explored in several recent works. Planning from computational tree logic was considered in [17], whereas in [13], [16], the authors focus on planning behavior of a team of robots from a single, global LTL specification. Fragments of LTL have been considered for vehicle routing problems for unmanned aerial vehicles in [11], and for search and rescue missions in [20]. A 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 agent.

In [9], 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. To cope with this issue, in this paper, we propose a receding horizon approach to multi-agent planning. The idea is to translate infinite horizon planning into an infinite sequence of finite horizon planning problems similarly as in [21], where the authors leverage the same idea 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], [18] for a single-agent LTL planning to achieve a locally optimal behavior. The contribution of this paper can be summarized as the introduction of an

efficient, limited horizon planning technique in the context of bottom-up control strategy synthesis for multi-agent systems from local LTL specifications. To our best knowledge, such an approach has not been taken to address the distributed multi-agent planning problem and its extreme computational demands before.

The rest of the paper is structured as follows. In Sec. II, we fix necessary preliminaries. Sec. III introduces the problem statement and summarizes our approach. In Sec. IV, the details of the solutions are provided. We present an illustrative example and simulation results in Sec. V, and we conclude and outline several directions for future research in Sec. VI.

II. Preliminaries

Let 2^{S}, and S^{ω} denote the set of all subsets of a set S, and the set of all infinite sequences of elements of S, respectively.

A. System Model and Specification

Definition 1 (Transition System) *A* labeled deterministic transition system (TS) *is a tuple* $\mathcal{T} = (S, s_{init}, R, \Pi, L)$, *where*

- S *is a finite set of states;*
- $s_{init} \in S$ *is the initial state;*
- $R \subseteq S \times S$ *is a deterministic transition relation;*
- Π *is a set of services;*
- $L : S \to 2^{\Pi}$ *is a labeling function.*

The labeling function assigns to each state s a subset of *services* that are available in that state. In other words, there is an option to provide or not to provide a service $\pi \in L(s)$ in the state s. In contrast, π cannot be provided in s, where $\pi \notin L(s)$. The transition system evolves as follows: from a current state, either a subset of available services is provided, or the system changes its state by executing a transition while providing a so-called *silent service* ε. Note, that we distinguish between an empty set of services $\emptyset$ and a silent service ε. Formally, a *trace* of $\mathcal{T}$ is an infinite alternating sequence of states and subsets of services $\tau = s_1\varpi_1 s_2\varpi_2 \ldots$, such that $s_1 = s_{init}$, and for all $i \geq 1$ either (i) $s_i = s_{i+1}$, and $\varpi_i \subseteq L(s_i)$, or (ii) $(s_i, s_{i+1}) \in R$, and $\varpi_i = \varepsilon$.

A trace $\tau = s_1\varpi_1 s_2\varpi_2 \ldots$ is associated with a sequence $w_{\varepsilon}(\tau) = \varpi_1\varpi_2 \ldots \in (2^{\Pi} \cup \{\varepsilon\})^{\omega}$, and the *word produced by* τ defined as the subsequence of the non-silent elements of $w_{\varepsilon}(\tau)$. Formally, a word produced by $\tau = s_1\varpi_1 s_2\varpi_2 \ldots$ is $w(\tau) = \varpi_{i_1}\varpi_{i_2} \ldots \in (2^{\Pi})^{\omega}$, such that $\varpi_1, \ldots, \varpi_{i_1-1} = \varepsilon$, $\varpi_{i_j+1}, \ldots, \varpi_{i_{j+1}-1} = \varepsilon$ and $\varpi_{i_j} \neq \varepsilon$, for all $j \geq 1$. The sequence of indexes $\mathbb{T}(\tau) = \mathbb{T}(w_{\varepsilon}(\tau)) = i_1 i_2 \ldots$ is the sequence of time instances, when non-silent services are provided, called a *service time sequence*. Note that the word $w(\tau)$ and the service time sequence $\mathbb{T}(\tau)$ might be finite as well as infinite. However, as in this work we are interested in infinite, recurrent behaviors, we will consider as *valid* traces only those producing infinite words.

Definition 2 *An LTL formula* ϕ *over the set of services* Π *is defined inductively as follows:*

1) *every service* $\pi \in \Pi$ *is a formula, and*
2) *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 formulas,*

where $\neg$ *(negation) and* $\vee$ *(disjunction) are standard Boolean connectives, and* X *(next),* U *(until),* F *(eventually), and* G *(always) are temporal operators.*

The semantics of LTL is defined over infinite words over 2^{Π}, such as those produced by traces of the TS from Def. 1 (see, e.g., [1] for details). Intuitively, π is satisfied on a word $w = w(1)w(2)w(3)\ldots$ if it holds at $w(1)$. Formula $\mathsf{X}\,\phi$ holds true if ϕ is satisfied on the word suffix $w(2)w(3)\ldots$, whereas $\phi_1 \,\mathsf{U}\, \phi_2$ states that ϕ_1 has to be true until ϕ_2 becomes true. Finally, $\mathsf{F}\,\phi$ and $\mathsf{G}\,\phi$ are true if ϕ holds on w eventually, and always, respectively.

The language of all words that are accepted by an LTL formula ϕ is denoted by $\mathcal{L}(\phi)$. A trace τ of $\mathcal{T}$ satisfies LTL formula ϕ, denoted by $\tau \models \phi$ iff the word $w(\tau)$ satisfies ϕ, denoted $w(\tau) \models \phi$.

Remark 1 *Traditionally, LTL is defined over the set of atomic propositions (APs) instead of services (see, e.g. [1]). In transition systems, the APs represent inherent properties of system states. The labeling function* L *then partitions APs into those that are true and false in each state. The LTL formulas are interpreted over runs, i.e., sequences of states of transition systems. Run* $s_1 s_2 \ldots$ *satisfies* ϕ *if and only if the* $L(s_1)L(s_2)\ldots \models \phi$.

In this work, we consider an alternative definition of LTL semantics to describe the desired tasks. Particlularly, we perceive atomic propositions as offered services rather than undetachable inherent properties of the system states. For instance, given that a state is determined by the physical location of an agent, we consider atomic propositions of form "in this location, data can be gathered", or "there is a recharger in this location" rather than "this location is dangerous". In other words, the agent is given the option to decide whether an atomic proposition $\pi \in L(s)$ *is in state* s *satisfied or not. In contrast,* $\pi \in \Pi$ *is never satisfied in state* s*, such that* $\pi \notin L(s)$*. The LTL specifications are then interpreted over sequences of executed services along traces instead of the words produced by the traces.*

B. Strategy Synthesis

Given a transition system $\mathcal{T}$ with the set of atomic propositions Π and an automaton $\mathcal{A}$ over 2^{Π}, we say that a trace τ of $\mathcal{T}$ *satisfies* $\mathcal{A}$, denoted by $\tau \models \mathcal{A}$ if and only if the word produced by τ belongs to the language of $\mathcal{A}$, i.e., if $w(\tau) \in L(\mathcal{A})$.

Definition 3 (Büchi Automaton) *A Büchi automaton (BA) is a tuple* $\mathcal{B} = (Q, q_{init}, \Sigma, \delta, F)$, *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;*
- F *is the acceptance condition.*

The semantics of Büchi automata are defined over infinite words over Σ, such as those generated by a transition system from Def. 1 if $\Sigma = 2^{\Pi}$. A *run* of the BA $\mathcal{B}$ *over* an input word $w = w(1)w(2)\ldots$ is a sequence $\rho = q_1 q_2 \ldots$, such that $q_1 = q_{init}$, and $(q_i, w(i), q_{i+1}) \in \delta$, for all $i \geq 1$. Word w is accepted if there exists an accepting run ρ over w that intersects F infinitely many times. $\mathcal{L}(\mathcal{B})$ is the *language* of all accepted words. Any LTL formula ϕ over Π can be translated into a BA $\mathcal{B}$, such that $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\phi)$ [1] using an off-the-shelf software tool, such as [8].

Given a BA $\mathcal{B}$, we define the set of states $\hat{\delta}^k(q)$ that are reachable from a state $q \in Q$ in exactly k steps inductively as (i) $\hat{\delta}^0(q) = \{q\}$, and (ii) $\hat{\delta}^{k+1}(q) = \bigcup_{q' \in \hat{\delta}^k(q)} \{q'' \mid \exists\, \sigma \in \Sigma.\, (q', \sigma, q'') \in \delta\}$, for all $k \geq 0$.

Definition 4 (Product Automaton) *A product of a transition system $\mathcal{T} = (S, s_{init}, R, \Pi, L)$ and a Büchi automaton $\mathcal{B} = (Q, q_{init}, 2^{\Pi}, \delta, F)$ is an automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B} = (Q_{\mathcal{P}}, q_{init,\mathcal{P}}, \Sigma_{\mathcal{P}}, \delta_{\mathcal{P}}, F_{\mathcal{P}})$, where*

- $Q_{\mathcal{P}} = S \times Q$;
- $q_{init,\mathcal{P}} = (s_{init}, q_{init})$;
- $\Sigma_{\mathcal{P}} = 2^{\Pi} \cup \{\varepsilon\}$;
- $((s,q), \sigma, (s', q')) \in \delta_{\mathcal{P}}$ *iff either*
  - $(s, s') \in R, \sigma = \varepsilon$, $q = q'$, *or*
  - $s = s', \sigma \subseteq L(s)$, *and* $(q, \sigma, q') \in \delta$;
- $F_{\mathcal{P}} = S \times F$.

A run of the product automaton over a word $w = \sigma_1\sigma_2 \ldots \in 2^{\Pi}$ is a sequence of states $\rho = p_1 p_2 \ldots$, where $p_1 = q_{init,\mathcal{P}}$, with the property that and there exists a word $w_\varepsilon = w_\varepsilon(1)w_\varepsilon(2)\ldots = \varepsilon \ldots \varepsilon\, \sigma_1 \varepsilon \ldots \varepsilon\, \sigma_2 \varepsilon \ldots \in \Sigma_{\mathcal{P}}^{\omega}$, such that $(p_i, w_\varepsilon(i), p_{i+1}) \in \delta_{\mathcal{P}}$, for all $i \geq 1$. Such a run is accepting if it intersects $F_{\mathcal{P}}$ infinitely many times.

An accepting run $\rho = (s_1, q_1)(s_2, q_2)\ldots$ over a word $w = \sigma_1\sigma_2 \ldots \in (2^{\Pi})^{\omega}$ of the product automaton projects onto a valid trace $\tau = s_1 \varpi_1 s_2 \varpi_2 \ldots$ of $\mathcal{T}$, which produces the word w. At the same time, $w \in \mathcal{L}(\mathcal{B})$. Dually, there exists an accepting run of $\mathcal{P}$ over each word $w \in \mathcal{L}(\mathcal{B})$ that is produced by a valid trace τ of $\mathcal{T}$.

An automaton $(Q, q_{init}, \Sigma, \delta, F)$, can be viewed as a graph (V, E) with the set of vertices $V = Q$ and the set of edges E given by the transition function δ in the expected way. Thus, the standard notation from graph theory can be applied: A *path* in an automaton is a finite sequence of states and transition labels $q_i \xrightarrow{\sigma_i} q_{i+1} \ldots q_{l-1} \xrightarrow{\sigma_{l-1}} q_l$, such that $(q_j, \sigma_j, q_{j+1}) \in \delta$, for all $i \leq j < l$. A path is *simple* if $q_j = q_{j'} \Rightarrow j = j'$, for all $i \leq j, j' \leq l$. A path $q_i \xrightarrow{\sigma_i} \ldots q_l \xrightarrow{\sigma_l} q_{l+1}$, where $q_i \ldots q_l$ is a simple path and $q_{l+1} = q_i$, is called a *cycle*.

Let $succ(q) = \{q' \mid \exists \sigma.\, (q, \sigma, q') \in \delta\}$ denote the set of successors of q. Furthermore, let $dist(q, q')$ denote the length of the shortest simple path that begins in q and ends in q', i.e., the minimal number of states in a sequence representing a path $q \ldots q'$. If no such path exists, then $dist(q, q') = \infty$. If $q = q'$, then $dist(q, q') = 0$. A *shortest* path from q to q' is a path minimizing $dist(q, q')$, and can be computed using, e.g., Dijkstra algorithm (see, e.g., [5] for details).

Given a product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{B}$, a valid trace of $\mathcal{T}$ satisfying the specification $\mathcal{B}$ can be generated by finding a simple path from $q_{init,\mathcal{P}}$ (a trace prefix) to an accepting state q_f and a cycle $q_f \xrightarrow{\sigma_i} \ldots \xrightarrow{\sigma_l} q_f$, which contains at least one non-silent $\sigma_j \in 2^{\Pi}$, for some $j \in \{i, \ldots, l\}$ (a periodically repeated trace suffix). Such a simple path and cycle can be found using efficient graph algorithms.

III. Problem Formulation and Approach

In this section, we formally state our problem of multi-agent planning from individual LTL specifications. We outline the straightforwards solution based on the control strategy synthesis method presented in Sec. II-B, and we discuss the drawbacks of this solution. Finally, to cope with these drawbacks, we suggest an alternative appraoch that is futher elaborated in details in Sec. IV.

A. Problem Statement

Let us consider N agents, (e.g., robots in a partitioned environment). Each agent is modeled as a finite transition system $\mathcal{T}_i = (S_i, s_{init,i}, R_i, \Pi_i, L_i)$, for all $i \in \{1, \ldots, N\}$. States of the transition system correspond to states of the agents (e.g., the robot's physical location in the regions of the environment) and the transitions between them correspond to the agent's capabilities to change the state (e.g., the ability of the robots to move between two regions of the environment). We assume that $(s, s) \in R_i$, for all $s \in S_i$, i.e., that any agent i can stay in its current state, and we assume that each state $s \in S_i$ is reachable from all states $s' \in S_i$, i.e., that any agent can return to a state where it already was in the past. We consider that the agents' transitions are synchronized in time; they are triggered at the same time instant and whenever a transition of one agent is triggered, then a transition of every other agent is triggered as well. Without loss of generality, we assume that $\Pi_i \cap \Pi_j = \emptyset$, for all $i \neq j \in \{1, \ldots, N\}$, and that the set of silent services is $\mathcal{E} = \{\varepsilon_i \mid i \in \{1, \ldots, N\}\}$.

Each agent is given an LTL task ϕ_i over $\mathbf{\Pi}_i = \bigcup_{j \in d(i)} \Pi_j$, for some $\{i\} \subseteq d(i) \subseteq \{1, \ldots, N\}$. Informally, the satisfaction of an agent's task depends on, and only on the behavior of the subset of agents $d(i)$, including the agent itself. Formula ϕ_i is interpreted over the traces $\tau_j = s_{j,1}\varpi_{j,1}s_{j,2}\varpi_{j,2}\ldots$ of transition systems $\mathcal{T}_j$, where $j \in d(i)$. More precisely, the agent i decides the satisfaction of the formula ϕ_i based on the word $w(\tau_i)$ it produces and on the services of agents $\mathcal{T}_j, j \in d(i)$ provided at the time instances $\mathbb{T}(\tau_i)$. In other words, the agent $\mathcal{T}_i$ observes and takes into consideration the other agents' services only at the time instances, when $\mathcal{T}_i$ provides a service (even an empty one) itself. Formally, let $\mathfrak{w}_\varepsilon = \varpi_1\varpi_2 \ldots \in (2^{\mathbf{\Pi}_i} \cup \mathcal{E})^{\omega}$, where $\varpi_k = \bigcup_{j \in d(i)} \varpi_{j,k}$ denote the sequence of (silent and non-silent) services associated with the set of traces $\mathfrak{T}_i = \{\tau_j \mid j \in d(i)\}$, and let $\mathbb{T}(\tau_i) = \mathbb{T}(w_\varepsilon(\tau_i)) = k_1 k_2 \ldots$.

The *word produced* by $\mathfrak{T}_\mathrm{i}$ is then a sequence

$$w(\mathfrak{T}_i) = w(\mathfrak{w}_\varepsilon) = \omega_{k_1}\omega_{k_2}\ldots, \quad \text{such that } \omega_{k_m} = \varpi_{k_m} \cap 2^{\mathbf{\Pi}_i}, \text{ for all } m \geq 1. \tag{1}$$

The set of traces $\mathfrak{T}_i$ is called valid if the word $w(\mathfrak{T}_i)$ is infinite, i.e. if τ_i is valid. The formula ϕ_i is satisfied on a valid set of traces $\mathfrak{T}_i$, if and only if $w(\mathfrak{T}_i) \models \phi_i$.

Example 1 *Consider transition systems $\mathcal{T}_1, \mathcal{T}_2$, with $\Pi_1 = \{a\}$, and $\Pi_2 = \{b\}$, and their tasks $\phi_1 = a \wedge \mathsf{X}(a \wedge b)$, $\phi_2 = b \wedge \mathsf{X}(b \wedge a)$. Note that both $1 \in d(2)$, and $2 \in d(1)$. For traces τ_1, $w_\varepsilon(\tau_1) = \{a\}\varepsilon\varepsilon\{a\}\{\}\varepsilon\ldots$, and τ_2, $w_\varepsilon(\tau_2) = \varepsilon\varepsilon\varepsilon\{b\}\{b\}\varepsilon\ldots$, formula ϕ_1 is satisfied, as the word produced by $\mathfrak{T}_1$ is $w(\mathfrak{T}_1) = \{a\}\{a,b\}\{b\}\ldots$. In contrast, ϕ_2 is not satisfied, because $w(\mathfrak{T}_2) = \{a,b\}\{b\}\ldots$. Both formulas are satisfied if $w_\varepsilon(\tau_1)$ changes to $\{a\}\varepsilon\varepsilon\varepsilon\{a\}\varepsilon\ldots$*

Problem 1 Given N *agents represented as transition systems $\mathcal{T}_i = (S_i, s_{init,i}, R_i, \Pi_i, L_i)$, and LTL formulas ϕ_i over $\mathbf{\Pi}_i = \bigcup_{j\in d(i)} \Pi_j$, for all $i \in \{1,\ldots,N\}$,* find *a trace τ_i of each $\mathcal{T}_i$, such that $\mathfrak{T}_i = \{\tau_j \mid j \in d(i)\}$ is valid and satisfies the specification ϕ_i, for all $i \in \{1,\ldots,N\}$.*

As each of the LTL formulas ϕ_i, $i \in \{1,\ldots,N\}$ over $\mathbf{\Pi}_i$ can be translated into a language equivalent Büchi automaton, we can pose the problem equivalently as:

Problem 2 Given N *agents represented as transition systems $\mathcal{T}_i = (S_i, s_{init,i}, R_i, \Pi_i, L_i)$, and Büchi automata $\mathcal{B}_i = (Q_i, q_{init,i}, \delta_i, \mathbf{\Sigma}_i = 2^{\mathbf{\Pi}_i}, F)$, for all $i \in \{1,\ldots,N\}$,* find *a trace τ_i of each $\mathcal{T}_i$, such that $\mathfrak{T}_i = \{\tau_j \mid j \in d(i)\}$ is valid and produces a word $w(\mathfrak{T}_i) \in \mathcal{L}(\mathcal{B}_i)$.*

B. Straightforward Centralized Solution

An immediate solution to the Prob. 2 can be obtained by a slight modification to the standard control strategy synthesis procedure for transition systems from LTL specification (see Sec. II-B). Roughly, the procedure solving Prob. 2 include (1) partitioning the set of agents into dependency classes similarly as in [9], by iterative application of the rule that if $j \in d(i)$, then $\mathcal{T}_j$ belongs to the same dependency class as $\mathcal{T}_i$; (2) for each dependency class $D = \{\mathcal{T}_{d_1}, \ldots, \mathcal{T}_{d_m}\}$, constructing a transition system $\mathcal{T}_D$ with the set of states $S = S_{d_1} \times \ldots \times S_{d_m}$ that represents the synchronized behavior of agents within the class; (3) building a Büchi automaton $\mathcal{B}_D$, which accepts all the sequences $\mathfrak{w}_\varepsilon = \varpi_1\varpi_2\ldots \in (2^{\mathbf{\Pi}_D} \cup \mathcal{E})^\omega$, such that the produced word $w(\mathfrak{w}_\varepsilon) \in (2^{\mathbf{\Pi}_D})^\omega$ (see Eq. 1) satisfies ϕ_i, for all $T_i \in D$; (4) constructing a product automaton $\mathcal{P}_D$ of $\mathcal{T}_D$ and $\mathcal{B}_D$; and (5) using graph algorithms to find an accepting run of $\mathcal{P}_D$ that projects onto valid traces of $\mathcal{T}_i$, and accepting runs of $\mathcal{B}_i$, for all $T_i \in D$.

The outlined procedure is correct and complete; a solution is found if one exists and it is indeed a solution to Prob. 1. However, it suffers from a rapid growth of the product automaton state space with the increasing number of agents, leading to extreme computational demands that make the approach infeasible in practice. Particularly, if the size of D is N, the product automaton $\mathcal{T}_D \otimes \mathcal{B}_D$ is $\mathcal{O}(\prod_{1\leq i\leq N} |\mathcal{T}_i|)$, which is approx. $|T|^N$.

C. Our Approach

In this work, we aim on reducing the high computational complexity of the straightforward solution. Our approach is to avoid the execution of an offline, centralized control strategy generation procedure and to decompose the strategy synthesis problem into short horizon planning problems that are solved online, upon the execution of the system, similarly as in model predictive control. As a starting point, we consider the problem definition from Prob. 2.

In the sequel, we present an iterative method to select a temporary goal state for each agent within a short horizon, and compute and execute a finite trace fragment leading to this goal state. We show, that under certain assumptions, the repetitive implementation of the outlined algorithm leads to provable satisfaction of the desired specifications. The solution leverages ideas from LTL control strategy synthesis and also the construction of intersection Büchi automata [1].

IV. Problem Solution

In this section, we provide details of the proposed solution to Prob. 2. First, we introduce the procedures that are executed in each iteration of the algorithm, followed by the summary of the overall method. Along the procedures presentations, two assumptions are imposed to ensure the correctness of the algorithm and we discuss how they can be relaxed towards the end of this section.

Besides the set of transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_N$, and the specification automata $\mathcal{B}_1, \ldots, \mathcal{B}_N$, the inputs to each iteration of the algorithm are:

- current states of $\mathcal{T}_1, \ldots, \mathcal{T}_N$, denoted by $\mathfrak{s}_1, \ldots, \mathfrak{s}_N$, initially equal to $s_{init,1}, \ldots, s_{init,N}$, respectively;
- current states of $\mathcal{B}_1, \ldots, \mathcal{B}_N$, denoted by $\mathfrak{q}_1, \ldots, \mathfrak{q}_N$, initially equal to $q_{init,1}, \ldots, q_{init,N}$, respectively;
- linear ordering $\prec$ over $\{1,\ldots,N\}$, initially arbitrary;
- a fixed horizon $h \in \mathbb{N}$, which, loosely speaking, determines the depth of planning in the Büchi automata;
- a fixed horizon $H \in \mathbb{N}$ which, loosely speaking, determines the depth of planning in the transition systems.

A. Intersection Büchi Automata

In each iteration of the algorithm, we construct local automata that represent the intersection of relevant Büchi automata up to a pre-defined horizon h. We label their states with values that, simply put, indicate the progress towards the satisfaction of the desired properties. Later on, these values are used to set local goals in the short horizon planning.

We partition the set of Büchi automata $\mathbf{\Phi} = \{\mathcal{B}_1, \ldots, \mathcal{B}_N\}$ into the smallest possible subsets $\Phi_1, \ldots, \Phi_M$, such that any transition of any $\mathcal{B}_i \in \Phi_\ell$ up to horizon h from the current state does not impose restrictions on the behavior of any agent $\mathcal{T}_j$ with the property that $\mathcal{B}_j \notin \Phi_\ell$. This partition corresponds to the current *necessary and sufficient dependency* between agents up to the horizon h, and can dynamically change over the time.

Definition 5 (Participating Services) *Formally, we call a set of services Π_j, $j \in d(i)$* participating *in $q \in Q_i$ if*

(i) $j = i$, *or*

(ii) *there exist $q' \in Q_i$, $\sigma \in \mathbf{\Sigma}_i$, and $\varsigma \subseteq \Pi_j$ such that $(q, \sigma, q') \in \delta$, and $(q, (\sigma \setminus \Pi_j) \cup \varsigma, q') \notin \delta$.*

Intuitively, a set of services Π_j is participating in q, if some transition leading from q imposes restrictions on the services provided by agent j.

Definition 6 (Alphabet up to Horizon h) *For a state $q \in Q_i$, we define the alphabet $\mathbf{\Sigma}_i^h$ of $\mathcal{B}_i$ up to the horizon h as $\mathbf{\Sigma}_i^h(q) = 2^{\mathbf{\Pi}_i^h(q)}$, where*

$$\mathbf{\Pi}_i^h(q) = \bigcup_{\substack{q' \in \hat{\delta}_i^k(q) \\ 0 \leq k \leq h}} \{\Pi_j \mid \Pi_j \text{ is a participating service in } q'\}.$$

Definition 7 (Dependency Equivalence and Partition) *Given that $\mathfrak{q}_1, \ldots, \mathfrak{q}_N$ are the respective current states of Büchi automata $\mathcal{B}_1, \ldots, \mathcal{B}_N$, the partition of the set of Büchi automata $\mathbf{\Phi}$ is induced by the dependency equivalence $\sim^h$ defined on $\mathbf{\Phi}$ as follows:*

- $\mathcal{B}_i \sim^h \mathcal{B}_i$
- *if there exists $\mathcal{B}_k$, such that $\mathcal{B}_i \sim^h \mathcal{B}_k$, and $\Pi_j \subseteq \mathbf{\Pi}_k^h(\mathfrak{q}_k)$ or $\Pi_k \subseteq \mathbf{\Pi}_j^h(\mathfrak{q}_j)$, then also $\mathcal{B}_i \sim^h \mathcal{B}_j$.*

The desired partition is then $\{\Phi_1, \ldots, \Phi_M\}$, with the property that $(\mathcal{B}_i \sim^h \mathcal{B}_j) \iff (\mathcal{B}_i \in \Phi_\ell \iff \mathcal{B}_j \in \Phi_\ell)$. We associate each subset of Büchi automata Φ_ℓ with the set of indexes I_ℓ, such that $\mathcal{B}_i \in \Phi_\ell \iff i \in I_\ell$.

Note, that planning within the horizon h can now be done separately for each Φ_ℓ. Thus, from now on, in the remainder of this section and Sec. IV-B and IV-C, let us concentrate on planning for a *dependency class* of agents and specifications given by $I_\ell = \{1_\ell, \ldots, n_\ell\}$, for a fixed ℓ.

We are now ready to define the Büchi automata intersection up to the horizon h, for $\Phi_\ell = \{\mathcal{B}_{1_\ell}, \ldots, \mathcal{B}_{n_\ell}\}$. Let $i_\ell \prec j_\ell$, for all $1 \leq i < j \leq n$. In other words, we assume, without loss of generality, that the automata in Φ_ℓ are ordered according to $\prec$.

Definition 8 (Intersection Automaton) *The intersection automaton of $\mathcal{B}_{1_\ell}, \ldots, \mathcal{B}_{n_\ell}$ up to horizon h is $\mathcal{A}^h = (Q_\mathcal{A}, q_{init,\mathcal{A}}, \mathbf{\Sigma}_\mathcal{A}, \delta_\mathcal{A}, F_\mathcal{A})$, where*

- *$Q_\mathcal{A} \subset Q_{1_\ell} \times \ldots \times Q_{n_\ell} \times \mathbb{N}$ is a finite set of states, generated as described below;*
- *$q_{init,\mathcal{A}} = (\mathfrak{q}_{1_\ell}, \ldots, \mathfrak{q}_{n_\ell}, 1)$;*
- *$\mathbf{\Sigma}_\mathcal{A} = \{\sigma \in 2^{\mathbf{\Pi}_\mathcal{A}} \mid \forall i_\ell \in I_\ell.\ \sigma \cap 2^{\mathbf{\Pi}_{i_\ell}} \neq \emptyset \Rightarrow \varepsilon_{i_\ell} \notin \sigma\}$, where $\mathbf{\Pi}_\mathcal{A} = \bigcup_{i \in I_\ell} \Pi_i \cup \bigcup_{i \in I_\ell} \{\varepsilon_i\}$;*
- *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*
    - $(q_{i_\ell}, \sigma \cap \mathbf{\Pi}_{i_\ell}, q'_{i_\ell}) \in \delta_{i_\ell}$, *or*
    - $q_{i_\ell} = q'_{i_\ell}$, *and* $\varepsilon_{i_\ell} \in \sigma$
  iii)
  $$k' = \begin{cases} k+1 & \text{if } q_{(k \bmod n)_\ell} \in F_{(k \bmod n)_\ell}, \\ k & \text{otherwise.} \end{cases}$$
  Finally, $Q_\mathcal{A} = \bigcup_{0 \leq j \leq h} Q_\mathcal{A}^j$ and $\delta_\mathcal{A} = \bigcup_{1 \leq j \leq h} \delta_\mathcal{A}^j$;
- *$F_\mathcal{A} = \{(q_{1_\ell}, \ldots, q_{n_\ell}, k) \in Q_\mathcal{A} \setminus \{q_{init,\mathcal{A}}\} \mid q_{(k \bmod n)_\ell} \in F_{(k \bmod n)_\ell}\}$.*

The intersection automaton is not a Büchi automaton as it does not exhibit infinite runs. However, it is an automaton that reads finite words and thus, it can be viewed as a graph. Through k, we remember which accepting states of which $\mathcal{B}_{i_\ell}$ have been visited on a run towards the respective state; accepting states of all $\mathcal{B}_{1_\ell}, \ldots, \mathcal{B}_{i_\ell}$ have been visited on each path from $q_{init,\mathcal{A}}$ to the state with $k = i + 1$. Thus, intuitively, the greater k translates to the greater progress towards satisfaction of the individual specifications ordered according to $\prec$.

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

Intuitively, this assumption captures that at least a state which ensures a progress towards the satisfaction of the highest-order specification $\mathcal{B}_{1_\ell}$ is present in $\mathcal{A}$. This allows us to identify local goal states in $\mathcal{T}_{1_\ell}, \ldots, \mathcal{T}_{n_\ell}$ in the following subsection. Without this assumption, we would not be able to distinguish between "profitable" and "profitless" transitions of agents with respect to $\mathbf{\Phi}$. We analyze conditions under which Assump. 1 can be violated and propose a solution to its relaxation in Sec. IV-F.1.

Definition 9 (Progressive Function for $\mathcal{A}$) *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 follows:*

$$V_\mathcal{A}(q) = \big(k, -\min_{q_f \in F_\mathcal{A}} dist(q, q_f)\big).$$

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 state q, such that $V_\mathcal{A}(q) = (k, -\infty)$ within the horizon h, and hence, we remove these from $\mathcal{A}$. From Assump. 1, we have that $V_\mathcal{A}(q_{init,\mathcal{A}}) = (1, d)$, where $d \neq -\infty$.

B. Product System

The intersection automaton and its progressive function allows us to define which services should be provided in order to make a progress towards satisfaction of the specification. The remaining step is to plan the transitions of the individual agents to reach states in which these services are offered. We do so through definition of a product system that captures the allowed behaviors (finite trace fragments) of agents from I_ℓ up to horizon H. The states of the product system are evaluated based on the progressive function of $\mathcal{A}$, to indicate their progress towards satisfaction of the formula.

Definition 10 (Product System) *The product system up to the horizon H of the agent transition systems $\mathcal{T}_{i_\ell}, i_\ell \in I_\ell$,*

and the intersection Büchi automaton $\mathcal{A}^h$ from Def. 8 is an automaton $\mathcal{P}^H = (Q_\mathcal{P}, q_{init,\mathcal{P}}, \mathbf{\Sigma}_\mathcal{P}, \delta_\mathcal{P})$, where

- $Q_\mathcal{P} \subset S_{1_\ell} \times \ldots \times S_{n_\ell} \times Q_\mathcal{A}$ *is a finite set of states, generated as described below;*
- $q_{init,\mathcal{P}} = (\mathfrak{s}_{1_\ell}, \ldots, \mathfrak{s}_{n_\ell}, q_{init,\mathcal{A}})$*;*
- $\mathbf{\Sigma}_\mathcal{P} = \mathbf{\Sigma}_\mathcal{A}$;
- *Let* $Q^0_\mathcal{P} = \{q_{init,\mathcal{P}}\}$.
  For all $1 \leq j \leq H$, $(s'_{1_\ell}, \ldots, s'_{n_\ell}, q') \in Q^j_\mathcal{P}$ *and* $\big((s_{1_\ell}, \ldots, s_{n_\ell}, q), \sigma, ((s'_{1_\ell}, \ldots s'_{n_\ell}, q')\big) \in \delta^j_\mathcal{P}$ *iff for all* $i \in \{1, \ldots, n\}$, *either* $s_{i_\ell} = s'_{i_\ell}, \sigma \cap \Pi_{i_\ell} \subseteq L(s_{i_\ell})$ *and* $(q, \sigma, q') \in \delta_\mathcal{A}$, *or* $(s_{i_\ell}, s'_{i_\ell}) \in R_{i_\ell}$, $\varepsilon_{i_\ell} \in \sigma$ *and* $(q, \sigma, q') \in \delta_\mathcal{A}$.

  Finally, $Q_\mathcal{P} = \bigcup_{0 \leq j \leq H} Q^j_\mathcal{P}$ *and* $\delta_\mathcal{P} = \bigcup_{1 \leq j \leq H} \delta^j_\mathcal{P}$.

The set of accepting states $F_\mathcal{P}$ is not significant for the further computations, hence we omit it from $\mathcal{P}^H$. The tuple $\mathcal{P}^H$ is an automaton and can be viewed as a graph (see Sec. II). A path $p = q_1 \xrightarrow{\sigma_1} q_2 \ldots q_{m-1} \xrightarrow{\sigma_{m-1}} q_m$ in $\mathcal{P}^H$, where $q_1 = q_{init,\mathcal{P}}$ can be projected onto a finite trace prefix $\tau_{i_\ell}(p)$ of each $\mathcal{T}_{i_\ell}$, $i_\ell \in I_\ell$ in the expected way: the j-th state of $\tau_{i_\ell}(p)$ is s_{i_ℓ} if the j-th state of p is $q_j = (s_{1_\ell}, \ldots, s_{n_\ell}, q_\mathcal{A})$, and the j-th set of services of $\tau_{i_\ell}(p)$ is $\sigma_j \cap (\Pi_{i_\ell} \cup \{\varepsilon_{i_\ell}\})$, for all $j \in \{1, \ldots m\}$, and $j \in \{1, \ldots m-1\}$, respectively. The path p can be naturally projected onto a finite run prefix of the intersection automaton $\mathcal{A}^h$ and onto finite run prefixes of individual Büchi automata $\mathcal{B}_{i_\ell}$, too. Particularly, the j-th state of the run prefix $\rho_\mathcal{A}(p)$ of $\mathcal{A}^h$ is $q_\mathcal{A} = (q_{\mathcal{A},1_\ell}, \ldots, q_{\mathcal{A},n_\ell}, k)$ if the j-th state of p is $q_j = (s_{1_\ell}, \ldots, s_{n_\ell}, q_\mathcal{A})$, for all $j \in \{1, \ldots, m\}$; the j-th state of the run prefix $\rho_{i_\ell}(p)$ of $\mathcal{B}_{i_\ell}$ is then the state $q_{\mathcal{A},i_\ell}$.

Definition 11 (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 $\mathcal{A}^h$ (Def., 9), i.e., for all $(s_{1_\ell}, \ldots, s_{n_\ell}, q) \in Q_\mathcal{P}$, $V_\mathcal{P}\big((s_{1_\ell}, \ldots, s_{n_\ell}, q)\big) = V_\mathcal{A}(q)$.*

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, such for all $q' \in Q_\mathcal{P}$, it holds $V_\mathcal{P}(q) \geq V_\mathcal{P}(q')$.*

C. Plan Synthesis

Given $\mathcal{P}^H$, we compute the local plan as the shortest path $p = q_1 \xrightarrow{\sigma_1} q_2 \ldots q_m \xrightarrow{\sigma_m} q_{max}$ from $q_1 = q_{init,\mathcal{P}}$ to q_{max}, such that $\sigma_k \cap \mathcal{T}_{1_\ell} \neq \emptyset$, for some $k \in \{1, \ldots, m\}$, and q_{max} is a maximally progressive state reachable through such a path. The path can be computed using efficient graph algorithms, in linear time with respect to the size of P^H. We assume that such a path exists and show how to relax the assumption further in Sec. IV-F.2.

Assumption 2 *Assume that in $\mathcal{P}^H$, there exists at least one progressive state q_p reachable through a finite path $q_{init,\mathcal{P}} \xrightarrow{\sigma_1} q_2 \ldots q_m \xrightarrow{\sigma_m} q_p$, such that $\sigma_k \cap \mathcal{T}_{1_\ell} \neq \emptyset$, for some $k \in \{1, \ldots, m\}$.*

The projection of the found path onto individual agent transition systems gives finite trace prefixes $\tau_{i_\ell}(p) = \mathfrak{s}_{i_\ell} \varpi_{i_\ell,1} s_{i_\ell,2} \ldots s_{i_\ell,m}$, to be followed by each agent $i_\ell \in I_\ell$. Furthermore, it is guaranteed that at least agent $\mathcal{T}_{1_\ell}$ will provide at least one non-silent service along its trace prefix.

D. Plan Execution

Finally, in each iteration the individual trace prefixes $\tau_i(p) = \mathfrak{s}_i \varpi_{i,1} s_{i,2} \varpi_{i,2} \ldots s_{i,m_i}$ computed in the previous steps are executed as follows. Each agent $\mathcal{T}_i$, $i \in \{1, \ldots, N\}$ provides the services $\varpi_{i,1} \in L(\mathfrak{s}_i)$, and executes the transition to the state $s_{i,2}$. At the same time, the current state of each Büchi automaton $\mathcal{B}_i$, $i \in \{1, \ldots, N\}$ is updated to the second state $q_{i,2}$ of the run prefix $\rho_i(p) = \mathfrak{q}_i q_{i,2} \ldots q_{i,m_i}$ obtained by the projection of p onto $\mathcal{B}_i$. If $q_{i,2} \in F_i$, then the ordering $\prec$ is also updated, in such a way that i becomes of the lowest order, i.e., $j, j' \prec i$ for all $j, j' \in \{1, \ldots, N\} \setminus \{i\}$, while maintaining the mutual ordering of j and j'. Loosely speaking, this change reflects that a progress towards the satisfaction of specification $\mathcal{B}_i$ has been made and in the following iteration, we focus on making progress towards the satisfaction of the remaining specifications.

Algorithm 1 Solution to Prob. 2

Input: Transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_N$; Büchi automata $\mathcal{B}_1, \ldots, \mathcal{B}_N$; horizons $h \in \mathbb{N}$, $H \in \mathbb{N}$.
Output: *system execution* $(\tau_1, \ldots, \tau_N, \rho_1, \ldots, \rho_N)$, where $\tau_1, \ldots, \tau_N$ are traces of $\mathcal{T}_1, \ldots, \mathcal{T}_N$, and $\rho_1, \ldots, \rho_N$ are runs of $\mathcal{B}_1, \ldots, \mathcal{B}_N$, respectively

$\prec := (1, \ldots, N)$; $\mathfrak{s}_i := s_{init,i}$ $\mathfrak{q}_i := q_{init,i}, \forall i \in \{1, \ldots, N\}$
while *true* **do**
  compute the partition $\{I_1, \ldots, I_M\}$ (Def. 7)
  for all $\ell \in \{1, \ldots, M\}$ **do**
    construct $\mathcal{A}^h$ (Def. 8)
    construct $\mathcal{P}^H$ (Def. 10)
    find a shortest path p to a max. progressive state in $\mathcal{P}^H$
  end for
  for all $i \in \{1, \ldots, N\}$, suppose that $\tau_i(p) = \mathfrak{s}_i \varpi_{i,1} s_{i,2} \ldots s_{i,m_i}, \rho_i(p) = \mathfrak{q}_i q_{i,2} \ldots, q_{i,m_i}$, **do**
    provide services $\varpi_{i,1} \in L(\mathfrak{s}_i)$
    $\mathfrak{s}_i := s_{i,2}$; $\mathfrak{q}_i := q_{i,2}$
    if $\mathfrak{q}_i \in F_i$ **then**
      reorder $\prec$, s.t. $j \prec i$, for all $j \in \{1, \ldots, N\} \setminus \{i\}$
    end if
  end for
end while

This step has finalized one iteration of the algorithm and at this point, the next iteration is to be performed, starting with building the intersection automaton in Sec. IV-A. The overall solution is summarized in Alg. 1.

E. Correctness

Lemma 1 *A system execution $(\tau_1, \ldots, \tau_N, \rho_1, \ldots, \rho_N)$ computed by Alg. 1 satisfies the following, for all $i \in \{1, \ldots, N\}$:*

(i) *given that $\rho_i = q_{i,1} q_{i,2} \ldots$, $i \in \{1, \ldots, N\}$, and $\mathbb{T}(\tau_i) = k_1 k_2 \ldots$, the sequence $\varrho_i = q_{i,k_1} q_{i,k_2} \ldots$ is a run of $\mathcal{B}_i$, and furthermore $q_{i,1} = \ldots = q_{i,k_1 - 1}$, and $q_{i,k_j+1} = \ldots = q_{i,k_{j-1}-1}$, for all $j \geq 1$.*
(ii) *τ_i is a valid trace of $\mathcal{T}_i$.*
(iii) *ρ_i contains infinitely many states $q_f \in F_i$.*

Proof: Let t be an arbitrary time instant, and let $\tau_1^t, \ldots, \tau_N^t, \varpi_1^t, \ldots, \varpi_N^t, \rho_1^t, \ldots, \rho_N^t$ denote the current states and provided services of $\mathcal{T}_1, \ldots, \mathcal{T}_N$, and the current states of $\mathcal{B}_1, \ldots, \mathcal{B}_N$ at time t, respectively. Then, directly from the constructions of the intersection automaton and the product, we have the following: for all i, it holds that $(\tau_i^t, \tau_i^{t+1}) \in R_i$. Furthermore, if $\varpi_i^t = \varepsilon_i$, then $q_i^t = q_i^{t+1}$. On the other hand, if $\varpi_i^t \neq \varepsilon$, then $q_i^{t+1} \in \delta(q_i^t, \bigcup_{j \in d(i)} \varpi_j^t)$.

Consider that i is the most prioritized agent at time t, i.e., that $i \prec j$, for all $j \in \{1, \ldots, N\}$. Let τ_i' and ρ_i' are the finite trace and run prefixes of $\mathcal{T}_i$, $\mathcal{B}_i$ computed by Alg. 1 on lines 7–9 at time t to a maximally progressive state q_{max} of $\mathcal{P}^H$. Then, intuitively, at time $t+1$, this state is also present in $\mathcal{P}^H$. If a plan is changed to reach q'_{max}, then q'_{max} is "more progressive" than q_{max}, and thus closer to reaching an accepting state of $\mathcal{B}_i$. Altogether, thanks to the Assump. 1 and Assump. 2, we can state that a state q_{max}, which projects onto an accepting state q_f of $\mathcal{B}_i$ is reached. At the same time, it is ensured that at least one non-silent service is provided by $\mathcal{T}_i$ on this path. Furthermore, lines 12–14 of Alg. 1 ensure, that each $i \in \{1, \ldots, N\}$ will repeatedly become the most prioritized. Putting everything together, we can conclude that the lemma holds. ∎

Corollary 1 *A system execution $(\tau_1, \ldots, \tau_N, \rho_1, \ldots, \rho_N)$ returned by Alg. 1 provides a solution to Prob. 2.*

F. Relaxing the Assumptions

1) Relaxing Assump. 1: Intuitively, Assump. 1 may be violated from two different reasons: First, if the selected horizon h is too short, and although $F_{\mathcal{A}} = \emptyset$, there exists $h' > h$, such that $F_{\mathcal{A}} \neq \emptyset$ in $\mathcal{A}^{h'}$. Second, if $F_{\mathcal{A}} = \emptyset$ even for $h \to \infty$, i.e., if a wrong step was executed in the past that lead to the infeasibility of the formula. We show, how to identify the reason of the assumption violation and propose an approach to its relaxation.

Consider that $\mathcal{A}^h$ is built according to Def. 8 and that $F_{\mathcal{A}} = \emptyset$. In short, we 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 Assump. 1 and thus is used in constructing $\mathcal{P}^H$, maintaining the remainder of the solution as described in Sec. IV-B and IV-C. 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. Intuitivelly, we "undo" the service, we pretend that it has not been provided and mark this service as forbidden in the specification automata. The backtracking procedure is roughly summarized in Alg. 2.

Remark 2 *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 transition system and each Büchi automaton, the backtracking procedure will ensure that eventually, the agents' trace prefixes will be found by Alg. 1 without any further backtracking. Intuitively, this happens in the worst-case after the backtracking procedure rules out all the possible wrong transitions of the agents (line 5).

2) Relaxing Assump. 2: 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 such a situation, we systematically extend the horizon H similarly as we extended h in the Büchi automaton. Eventually, a progressive state will be found.

Algorithm 2 Backtracking

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

Output: Updates to Büchi automata $\mathcal{B}_1, \ldots, \mathcal{B}_N$

1: $k := \mathrm{t}$
2: **while** plan not found **do**
3: $k := k - 1$
4: Check, if the execution of $\bigcup_{i \in \{1,\ldots,N\}} \varpi_{i,k}$ can lead to a different set of states of Büchi automata than to $q_{1,t}, \ldots, q_{N,t}$. If so, apply the change and goto line 6.
5: Forbid the execution of $\bigcup_{i \in \{1,\ldots,N\}} \varpi_{i,k}$ 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, line 3–16, from $\mathfrak{s}_1 = s_{1,k}, \ldots, \mathfrak{s}_N = s_{N,k}, \mathfrak{q}_1 = q_{1,k}, \ldots, \mathfrak{q}_N = q_{N,k}$
7: If a plan was found in line 5, continue with execution of Alg. 1, otherwise goto line 2 of Backtracking.
8: **end while**

Remark 3 *Note, that Assump. 1 and 2 can be enforced by the selection 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 centralized solution discussed in Sec. III-B.*

V. Example

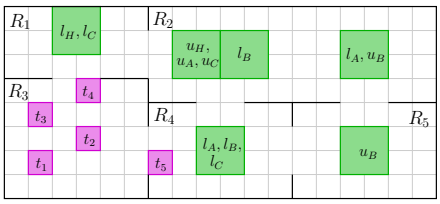
To demonstrate our approach and its benefits, we present an illustrative example of three mobile robots operating in a common workspace depicted in Fig. 1.(A). The agents can transit in between the adjacent cells of the partitioned environment and they can each provide various services. Agent 1 can load (l_H, l_A, l_B, l_C), carry, and unload (u_H, u_A, u_B, u_C) a heavy object H or a light object A, B, C. Agent 2 is capable of helping the agent 1 to load object 1 (h_H), and to execute simple tasks in the purple regions $(t_1 - t_5)$. Agent 3 is capable of taking a snapshot of the rooms $R_1 - R_5$ when being present within the respective room $(s_1 - s_5)$.

The robots are assigned complex tasks that require collaboration. Agent 1 would like agent 2 to help loading the heavy object. Then, it should carry the object to tje unloading point and unload it. After that, its task is to periodically load

and unload all the light objects. The goal of agent 2 is to periodically execute the sequence of 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 room R_4 at the moment of the execution. Finally, the goal of agent 3 is to patrol rooms R_2, R_4, R_5. The LTL formulas for the agents are: $\phi_1 = \mathsf{F}(l_H \wedge h_H \wedge \mathsf{X}\, u_H \wedge \bigwedge_{i\in\{A,B,C\}} \mathsf{GF}\,(l_i \wedge \mathsf{X} u_i))$, $\phi_2 = \mathsf{GF}\ (t_1 \wedge \mathsf{X}\ (t_2 \wedge \mathsf{X}\ (t_3 \wedge \mathsf{X}\ (t_4 \wedge \mathsf{X}\ t_5 \wedge s_4))))$, and $\phi_3 = \bigwedge_{i\in\{2,4,5\}} \mathsf{GF}\, s_i$.

We have implemented the proposed solution in MATLAB, and we illustrate the resulting trace prefixes after 40 iterations in Fig. 1.(B). 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 $H = 5$. The latter value was sometimes too low to find a solution, thus, in several cases it has been extended as described in IV-F. 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 are depicted in Fig. 2. In the centralized solution, all three agents belong to the dependency class, and hence, their synchronized product transition system has $144^3 \approx 3$ million states. In contrast, in our solution, the decomposition into dependency classes is done locally, and at most two agents belong to the same dependency class at the time (in iterations 1-5, and 17-31), resulting into product system sizes in order of thousands states. When the agents are not dependent on each other within h (in iterations 6-16, 32-40), the sizes of product systems are tens to hundreds states.

(A)



(B)

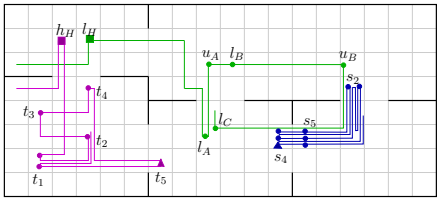


Fig. 1: (A) An example of an environment partitioned into cells. The environment consists of rooms $R_1, \ldots, R_5$. Green regions are loading and unloading points for a heavy object H and light objects A, B, C. Purple regions depict those where simple tasks $t_1, \ldots, t_5$ can be executed. (B) Traces of agent 1 (green), agent 2 (purple), and agent 3 (blue) after 40 iterations of Alg. 1. 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 cell labeled with s_4, respectively. Services l_H and h_H, and t_5 and s_4 are provided at the same time, illustrated as squares, and triangles, respectively. The rest of the provided services are depicted as circles.

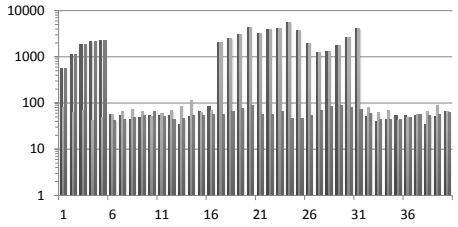


Fig. 2: The sizes of product automata in time (in logarithmic scale). The horizontal axis is labeled with the algorithm iteration number, the vertical one with the number of states of the product systems.

VI. 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 a finite horizon planning problems that are solved iteratively. Such solution brings two major advantages over the offline, centralized solution: First, the limited planning horizon enables each agent to restrict its focus only on those agents, that are constrained by its formula within the limited horizon, not within the whole infinite horizon. Thus, we reach a partially decentralized solution. Second, we reduce the size of handled state space.

Future research directions include involving various optimality requirements. Another aspect that we would like to address is robustness to small perturbations; an offline planning procedure with deterministic transition systems is not suitable for such problems and the complexity of planning with non-deterministic system is unbearable.

References

[1] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[2] C. Belta and L. C. G. J. M. Habets. Control of a class of nonlinear systems on rectangles. *IEEE Transactions on Automatic Control*, 51(11):1749–1759, 2006.

[3] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi. Motion planning with complex goals. *Robotics Automation Magazine, IEEE*, 18(3):55 –64, 2011.

[4] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, 2012.

[5] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[6] X. C. Ding, M. Lazar, and C. Belta. Receding horizon temporal logic control for finite deterministic systems. In *Proceedings of the American Control Conference (ACC)*, pages 715–720, 2012.

[7] I. Filippidis, D.V. Dimarogonas, and K.J. Kyriakopoulos. Decentralized multi-agent control from local LTL specifications. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 6235–6240, 2012.

[8] Paul Gastin and Denis Oddoux. LTL2BA tool, viewed September 2012. URL: http://www.lsv.ens-cachan.fr/ gastin/ltl2ba/.

[9] M. Guo and D. V. Dimarogonas. Reconfiguration in motion planning of single- and multi-agent systems under infeasible local LTL specifications. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 2758–2763, 2013.

[10] G. Jing, C. Finucane, V. Raman, and H. Kress-Gazit. Correct high-level robot control from structured english. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3543–3544, 2012.

[11] S. Karaman and E. Frazzoli. Vehicle routing with temporal logic specifications: Applications to multi-UAV mission planning. *International Journal of Robust and Nonlinear Control*, 21:1372–1395, 2011.

[12] M. Kloetzer and C. Belta. A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

[13] M. Kloetzer, X. C. Ding, and C. Belta. Multi-robot deployment from LTL specifications with reduced communication. In *Proceedings of the IEEE Conference on Decision and Control and European Control Conference (CDC/ECC)*, pages 4867–4872, 2011.

[14] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal Logic-based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[15] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[16] S. G. Loizou and K. J. Kyriakopoulos. Automated planning of motion tasks for multi-robot systems. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 78–83, 2005.

[17] M.M. Quottrup, T. Bak, and R.I. Zamanabadi. Multi-robot planning : a timed automata approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4417–4422, 2004.

[18] M. Svorenova, J. Tumova, J. Barnat, and I. Cerna. Attraction-based receding horizon path planning with temporal logic constraints. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 6749–6754, 2012.

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

[20] C. Wiltsche, F. A. Ramponi, and J. Lygeros. Synthesis of an asynchronous communication protocol for search and rescue robots. In *Proceedings of the European Control Conference (ECC)*, pages 1256–1261, 2013.

[21] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding Horizon Control for Temporal Logic Specifications. In *Hybrid systems: Computation and Control (HSCC)*, pages 101–110, 2010.