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], [22]: 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 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], [22] 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 in [11], and for search and rescue missions in [21]. A decentralized control of a robotic team from local LTL specification with communication constraints is proposed in [7]. Unlike in this paper, the specifications there are truly local and the agents do not impose any requirements on the other agents’ behavior. In [4], [20], 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. 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.

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 in Sec. V, and we conclude in Sec. VI. Due to space constraints, some proofs and discussions have been omitted and can be found in the technical report [19].
II. Preliminaries

Let $2^S$, and $S^\omega$ denote the set of all subsets of a set $S$, and the set of all infinite sequences of elements of $S$, respectively.

**Definition 1 (Transition System)** A labeled deterministic transition system (TS) is a tuple $T = (S, s_{\text{init}}, R, \Pi, L)$, where $S$ is a finite set of states; $s_{\text{init}} \in S$ is the initial state; $R \subseteq S \times S$ is a deterministic transition relation; $\Pi$ 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, $\pi$ 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 $\varepsilon$. Note, that we distinguish between an empty set of services $\emptyset$ and a silent service $\varepsilon$. Formally, a *trace* of $T$ is an alternating sequence of states and subsets of services $\tau = s_1 \omega \sigma_2 \sigma_2 \omega \ldots$, such that $s_1 = s_{\text{init}}$, and for all $i \geq 1$ either (i) $s_i = s_{i+1}$, and $\omega_i \subseteq L(s_i)$, or (ii) $(s_i, s_{i+1}) \in R$, and $\omega_i = \varepsilon$.

A trace $\tau = s_1 \omega \sigma_2 \omega \ldots$ is associated with a sequence $w_\tau(\tau) = \omega_1 \omega_2 \ldots \in (2^\Pi \cup \{\varepsilon\})^\omega$, and the word produced by $\tau$ defined as the subsequence of the non-silent elements of $w_\tau(\tau)$. Formally, a word produced by $\tau = s_1 \omega_1 \sigma_2 \omega_2 \ldots$ is $w(\tau) = \omega_1 \omega_2 \ldots \in (2^\Pi)^\omega$, such that $\omega_1, \ldots, \omega_{i-1} = \varepsilon, \omega_{i+1}, \ldots, \omega_{j+1} = \varepsilon$ and $\omega_i \neq \varepsilon$, for all $j \geq 1$.

The sequence of indexes $\Pi(\tau) = \Pi(w(\tau)) = i_1 i_2 i_3 \ldots$ is the sequence of time instances, when non-silent services are provided, called a *service time sequence*. 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 $\phi$ over the set of services $\Pi$ is defined inductively as follows:

1) every service $\pi \in \Pi$ is a formula, and
2) if $\phi_1$ and $\phi_2$ are formulas, then $\phi_1 \lor \phi_2$, $\neg \phi_1$, $X \phi_1$, $\phi_1 \lor \phi_2$, $F \phi_1$, and $G \phi_1$ are each formulas, where $\neg$ (negation) and $\lor$ (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 $2^\Pi$, such as those produced by traces of the TS from Def. 1 (see, e.g., [1] for details). Intuitively, $\pi$ is satisfied on a word $w = w(1)w(2)w(3)\ldots$ if it holds at $w(1)$. Formula $X \phi$ holds true if $\phi$ is satisfied on the word suffix $w(2)w(3)\ldots$, whereas $\phi_1 \lor \phi_2$ states that $\phi_1$ has to be true until $\phi_2$ becomes true. Finally, $F \phi$ and $G \phi$ are true if $\phi$ holds on $w$ eventually, and always, respectively. The language of all words that are accepted by an LTL formula $\phi$ is denoted by $L(\phi)$. A trace $\tau$ of $T$ satisfies LTL formula $\phi$, denoted by $\tau \models \phi$ iff the word $w(\tau)$ satisfies $\phi$, denoted $w(\tau) \models \phi$.

**Definition 3 (Büchi Automaton)** A Büchi automaton (BA) is a tuple $B = (Q, q_{\text{init}}, \Sigma, \delta, F)$, where $Q$ is a finite set of states; $q_{\text{init}} \in Q$ is the initial state; $\Sigma$ is an input alphabet; $\delta \subseteq Q \times \Sigma \times Q$ is a non-deterministic transition relation; $F$ is the acceptance condition.

The semantics of Büchi automata are defined over infinite words over $\Sigma$, such as those generated by a transition system from Def. 1 if $\Sigma = 2^\Pi$. A *run* of the BA $B$ over an input word $w = w(1)w(2)\ldots$ is a sequence $q = q_1 q_2 \ldots$, such that $q_1 = q_{\text{init}}$, and $(q_i, w(i), q_{i+1}) \in \delta$, for all $i \geq 1$.

A word $w$ is accepted if there exists an accepting run $\rho$ over $w$ that intersects $F$ infinitely many times. $L(B)$ is the language of all accepted words. Any LTL formula $\phi$ over $\Pi$ can be translated into a BA $B$, such that $L(B) = L(\phi)$ using an off-the-shelf software tool, such as [8].

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

An automaton $(Q, q_{\text{init}}, \Sigma, \delta, F)$, 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 $\delta$ 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 $q_i \xrightarrow{\sigma_i} q_{i+1} \ldots q_{i-1} \xrightarrow{\sigma_{i-1}} q_i$, such that $(q_j, q_{j+1}) \in \delta$, 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 $q_i \xrightarrow{\sigma_i} q_{i+1} \ldots q_l \xrightarrow{\sigma_l} q_{l+1}$, where $q_i \ldots q_l$ is a simple path and $q_{l+1} = q_i$, is called a cycle.

Let $\text{succ}(q) = \{q' \mid \exists \sigma(q, q', q'' \in \delta\}$ denote the set of successors of $q$. Furthermore, let $\text{dist}(q, q')$ 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 $\text{dist}(q, q') = \infty$.

If $q = q'$, then $\text{dist}(q, q') = 0$. A shortest path from $q$ to $q'$ is a path minimizing $\text{dist}(q, q')$, and can be computed using, e.g., Dijkstra algorithm (see, e.g., [5] for details).

**III. Problem Formulation and Approach**

In this section, we formally state our problem of multi-agent planning from individual LTL specifications.

Let us consider $N$ agents, (e.g., robots in a partitioned environment). Each agent is modeled as a finite transition system $T_i = (S_i, s_{\text{init}}, R_i, \Pi_i, L_i)$, 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 $E = \{e_i \mid i \in \{1, \ldots, N\}\}$.

Each agent is given an LTL task $\phi_i$ over $\Pi_i = \bigcup_{\text{dist}(i)} \Pi_j$, 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 $\phi_i$ is interpreted over the traces $\tau_j$ ... up to the horizon $h$ as
$\sum_{h}^{i(q)} = 2 \Pi_{h}^{i(q)}$, where
$\Pi_{h}^{i(q)} = \bigcup_{q' \in \hat{\delta}_k^{i(q)}} \{ \Pi_j \mid \Pi_j \text{ is participating in } q' \}$.

As a starting point, we consider Prob. 2. We suggest 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 the satisfaction of the desired
tasks. The solution leverages ideas from LTL control synthesis
and the construction of Büchi automata intersection [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 $T_1, \ldots, T_N$, and
the specification automata $B_1, \ldots, B_N$, the inputs to each
iteration of the algorithm are:

- current states of $T_1, \ldots, T_N$, denoted by $s_1, \ldots, s_N$,
  initially equal to $s_{\text{init}, 1}, \ldots, s_{\text{init}, N}$, respectively;
- current states of $B_1, \ldots, B_N$, denoted by $q_1, \ldots, q_N$,
  initially equal to $q_{\text{init}, 1}, \ldots, q_{\text{init}, N}$, 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 $\Phi = \{B_1, \ldots, B_N\}$
to the smallest possible subsets $\Phi_1, \ldots, \Phi_M$, such that
any transition of any $B_i \in \Phi_i$ up to horizon $h$ from the
current state does not impose restrictions on the behavior
of any agent $T_j$ with the property that $B_j \not\in \Phi_i$. 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 4 (Participating Services) Formally, we call a
set of services $\Pi_j$, $j \in d(i)$ participating in $q \in Q_i$ if
(i) $j = i$, or (ii) there exist $q' \in Q_i$, $\sigma \in \Sigma_i$, and $\varsigma \subseteq \Pi_j$ such that $(q, \sigma, q') \in \delta$, and $(q, (\sigma \setminus \Pi_j) \cup \varsigma, q') \not\in \delta$.

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

Definition 5 (Alphabet up to Horizon $h$) For a state $q \in Q_i$, we define the alphabet $\Sigma_i^{h}(q)$ of $B_i$ up to the horizon $h$ as
$\Sigma_i^{h}(q) = 2\Pi_i^{h}(q)$, where
$\Pi_i^{h}(q) = \bigcup_{q' \in \delta_i^{h}(q)} \{ \Pi_j \mid \Pi_j \text{ is participating in } q' \}$. 

1777
Definition 6 (Dependency Equivalence and Partition)

Given that \( q_1, \ldots, q_N \) are the respective current states of \( \text{B"uchi automata } B_1, \ldots, B_N \), the partition of the set of B"uchi automata \( \Phi \) is induced by the dependency equivalence \( \sim^h \) defined on \( \Phi \) as follows: (i) \( B_i \sim^h B_i \), and (ii) if there exists \( B_k \) such that \( B_i \sim^h B_k \), and \( \Pi \subseteq \Pi_k(q_k) \) or \( \Pi_k \subseteq \Pi_i(q_i) \), then also \( B_i \sim^h B_j \). The desired partition is then \( \{ B_1, \ldots, B_M \} \), such that the \( j \)-th state of \( B_i \), \( B_j \), \( B_k \), \( B_l \), respectively, are \( \Pi \)-equivalent.

Note, that planning within the horizon \( h \) can now be done separately for each \( \Phi_i \). Thus, from now on, in the remainder of this section and Sec. IV-B and IV-C, let us concentrate on the dependency class of agents and specifications generated by \( I_i = \{ 1, \ldots, n_i \} \), for a fixed \( i \).

We are now ready to define the B"uchi automata inter-connection up to the horizon \( h \), for \( \Phi_i = \{ B_{1i}, \ldots, B_{ni} \} \). Let \( i \tau < j \tau \), for all \( 1 \leq i < j \leq n \). In other words, we assume, without loss of generality, that the automata in \( \Phi_i \) are ordered according to \( < \).

Definition 7 (Intersection Automaton)

The intersection automaton of \( B_{1i}, \ldots, B_{ni} \), up to horizon \( h \), is \( A^h = (Q_A, \delta, \Pi_A, \delta_A, F_A) \), where

- \( Q_A \subset Q_{1i} \times \cdots \times Q_{ni} \times \mathbb{N} \) is a finite set of states, generated as described below;
- \( q_{init,A} = (q_{1i,1}, \ldots, q_{ni,1}) \); 
- \( \Pi_A = \{ \sigma \in \Pi_A \mid q_{init,A}, \sigma \in Q_A \} \); 
- \( Q^0_A = \{ q_{init,A} \} \).

For all \( 1 \leq j \leq h \), we define \( (q_{1i,j}, \ldots, q_{ni,j}, k) \in Q^j_A \) and \( ((q_{1i,j}, \ldots, q_{ni,j}, k), (q_{1i,j}, \ldots, q_{ni,j}, k')) \in \delta_A \iff \)

i) \( q_{1i,j}, \ldots, q_{ni,j}, k \in Q^j_A \),

ii) for all \( i \) in \( I_i \), either \( q_{1i,j}, \ldots, q_{ni,j} \in \Pi_i \), or \( q_{1i,j}, \ldots, q_{ni,j} \in \delta_i \), or \( q_{1i,j}, \ldots, q_{ni,j} \in \Pi_i \), and \( \delta_i \), or \( k' = k + 1 \)

iii) \( k' \) \( \in \{ k+1 \} \) if \( q_{init} \in F \).

Finally, \( Q_A = \bigcup_{0 \leq j \leq h} Q^j_A \) and \( \delta_A = \bigcup_{1 \leq j \leq h} \delta^j_A \); 
- \( F_A = \{ (q_{1i,j}, \ldots, q_{ni,j}) \in Q_A \mid q_{init} \in F \} \).

The intersection automaton is not a B"uchi 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 \( B_i \) have been visited on a run towards the respective state; accepting states of all \( B_{1i}, \ldots, B_{ki} \) have been visited on each path from \( q_{init,A} \) 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 \( < \).

Assumption 1 Assume that \( F_A \) is not empty.

Intuitively, this assumption captures that at least a state which ensures a progress towards the satisfaction of the highest-order specification \( B_{1i} \) is present in \( A \). This allows us to identify local goal states in \( T_{1i}, \ldots, T_{ni} \) in the following subsection. Without this assumption, we would not be able to distinguish between “profitable” and “profitless” transitions of agents with respect to \( \Phi \). We analyze conditions under which Assump. 1 can be violated and propose a solution to its relaxation in the technical report [19].

Definition 8 (Progressive Function for \( A \)) The progressive function \( V_A : Q_A \to \mathbb{N} \) is for a state \( q = (q_{1i}, \ldots, q_{ni}, k) \) defined as follows:

\[
V_A(q) = (k, - \min_{q \in F} \text{dist}(q, q_f)).
\]

The increasing value of \( V_A \) indicates a progress towards the satisfaction of the individual local specifications in \( \Phi_i \), ordered according to \( < \). No progress can be achieved from state \( q \), such that \( V_A(q) = (k, -\infty) \) within the horizon \( h \), and hence, we remove these from \( A \). From Assump. 1, we have that \( V_A(q_{init,A}) = (1, d) \), where \( d \neq -\infty \).

B. Product System

The intersection automaton and its progressive function allow 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 \( I_i \) up to horizon \( H \). The states of the product system are evaluated based on the progressive function of \( A \), to indicate their progress towards satisfaction of the formula.

Definition 9 (Product System) The product system up to the horizon \( H \) of the agent transition systems \( T_{1i}, k \) in \( I_i \), and the intersection B"uchi automaton \( A^h \) from Def. 7 is an automaton \( P^H = (Q_P, q_{init,p}, \sigma, \delta_P) \), where

- \( Q_P \subset S_{1i} \times \cdots \times S_{ni} \times Q_A \) is a finite set of states, generated as described below;
- \( q_{init,p} = (s_{1i}, \ldots, s_{ni}, q_{init,A}) \); 
- \( \sigma = \Pi_A \); 
- \( Q^0_P = \{ q_{init,p} \} \).

For all \( 1 \leq j \leq H \), \( (s_{1i,j}, \ldots, s_{ni,j}, q) \in Q^j_P \) and \( ((s_{1i,j}, \ldots, s_{ni,j}, q_1), \sigma, (s_{1i,j}, \ldots, s_{ni,j}, q_2), \sigma) \in \delta_P \) iff for all \( i \in \{ 1, \ldots, n \} \), \( s_{1i,j} \notin \bullet \Pi_{1i} \subseteq L(s_{1i,j}), \sigma, (s_{1i,j}, \ldots, s_{ni,j}, q') \in \delta_P \), or \( (s_{1i,j}, \ldots, s_{ni,j}, q') \in \delta_P, \sigma, (s_{1i,j}, \ldots, s_{ni,j}, q') \in \delta_P \).

Finally, \( Q_P = \bigcup_{0 \leq j \leq H} Q^j_P \) and \( \delta_P = \bigcup_{1 \leq j \leq H} \delta^j_P \).

The set of accepting states \( F_P \) is not significant for the further computations, hence we omit it from \( P^H \). The tuple \( P^H \) is an automaton and can be viewed as a graph (see Sec. II). A path \( p = q_1 \sigma_1 \rightarrow q_2 \sigma_2 \cdots q_{m-1} \sigma_{m-1} \rightarrow q_m \) in \( P^H \), where \( q_1 \) and \( q \) can be projected onto a finite trace prefix \( \pi(q) \) of each \( T_{1i}, k \) in the expected way: the \( j \)-th state of \( \pi(q) \) is \( s_{1i,j} \), the \( j \)-th state of \( q \) is \( q_j = (s_{1i,j}, \ldots, s_{ni,j}, q_A) \), and the \( j \)-th set of services of \( \pi(q) \) is \( \Pi_f \), \( \sigma_f \), \( \sigma_f \) 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 \( A^h \) and onto finite run prefixes of individual B"uchi automata \( B_{1i} \), too. Particularly, the \( j \)-th state

\[1778\]
of the run prefix $\rho_i(p)$ of $A^h$ is $q_A = (q_{A,1}, \ldots, q_{A,n}, k)$ if the $j$-th state of $p$ is $q_j = (s_{1}, \ldots, s_{n}, q_A)$, for all $j \in \{1, \ldots, m\}$; the $j$-th state of the run prefix $\rho_i(p)$ of $B_i$ is then the state $q_{A,i}$.  

**Definition 10 (Progressive Function and State)** The progressive function $V_P : Q_P \to \mathbb{N}_0 \times \mathbb{Z}_n$ is inherited from the intersection automaton $A^h$ (Def. 8), i.e., for all $(s_{1}, \ldots, s_{n}, q) \in Q_P$, $V_P((s_{1}, \ldots, s_{n}, q)) = V_A(q)$. A state $q \in Q_P$ is a progressive state if $V_P(q) > V_P(q')$. A maximally progressive state is $q_A = (q_{A,1}^{\ell}, \ldots, q_{A,n}^{\ell}, k)$, if the $j$-th state of $p$ is $q_j = (s_{1}^{\ell}, \ldots, s_{n}^{\ell}, q_A)$, for all $j \in \{1, \ldots, m\}$.

A state $\tau \in T_i$ is in the satisfaction of the remaining specifications.

**Correctness**

The overall solution is summarized in Alg. 1.

**Algorithm 1 Solution to Prob. 2**

**Input:** Transition systems $T_1, \ldots, T_N$: Büchi automata $B_1, \ldots, B_N$: 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 $T_1, \ldots, T_N$, and $\rho_1, \ldots, \rho_N$ are runs of $B_1, \ldots, B_N$, respectively.

1: $\prec := (1, \ldots, N)$; $s_i := s_{\text{init},i}$; $q_i := s_{\text{init},i}$, $\forall i \in \{1, \ldots, N\}$
2: while true do
3: compute the partition $\{I_1, \ldots, I_M\}$ (Def. 6)
4: for all $i \in \{1, \ldots, M\}$ do
5: construct $A^h$ (Def. 7)
6: construct $P^H$ (Def. 9)
7: find a shortest path $p$ to a maximal progressive state in $P^H$
8: end for
9: for all $i \in \{1, \ldots, N\}$, suppose that $\tau_i(p) = s_i, s_{i,1}, s_{i,2}, \ldots, s_{i,m_i}, \rho_i(p) = q_i, q_{i,1}, \ldots, q_{i,m_i}$, do
10: provide services $\varpi, \tau_i \subseteq L(s_i)$
11: $s_i := s_{i,1}$; $q_i := q_{i,2}$
12: if $q_{i} \in F_i$ then
13: reorder $\prec$, s.t. $j < i$, for all $j \in \{1, \ldots, N\} \setminus \{i\}$
14: end if
15: end for
16: end while

(i) given that $\rho_i = q_i, q_{i,1}, \ldots, \in \{1, \ldots, N\}$, and $T(\tau_i) = b_1, b_2, \ldots$ the sequence $q_i = q_i, k_1, q_{i,2}, \ldots$ is a run of $B_i$, and furthermore $q_i, k_1, \ldots = q_{i,1}$, and $q_{i, k_j} = \ldots = q_{i, k_j-1}$, for all $j \geq 1$.

(ii) $\tau_i$ is a valid trace of $T_i$.

(iii) $\rho_i$ contains infinitely many states $q_i \in F_i$.

See the technical report [19] for the proof.

**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.

**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 $R_1 - R_5$ while 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 the 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: $\phi_1 = F(l_H \land l_H \land X u_H \land \bigwedge_{i \in \{A,B,C\}}GF (l_i \land X u_i))$, $\phi_2 = GF (t_1 \land X (t_2 \land X (t_3 \land X (t_4 \land X t_5 \land s_4))))$, and $\phi_3 = \bigwedge_{i \in \{2,4,5\}}GF s_i$.  

1779
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 [19]. The maximum value needed in order to find a solution was $H = 9$. In the centralized solution, all three agents belong to the dependency class, and hence, their synchronized product transition system has $143^3 \approx 3$ 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.

![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 $s_A$, respectively. Services $l_H$ and $h_H$, and $t_5$ and $s_A$ are provided at the same time, illustrated as squares, and triangles, respectively. The rest of the provided services are depicted as circles.](image)

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 problem 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, or addressing robustness to small perturbations.

REFERENCES