# Multi-Agent Strategy Synthesis for LTL Specifications through Assumption Composition

Georg Friedrich Schuppe and Jana Tumova*

***Abstract*— We propose a compositional solution to the strategy synthesis problem for LTL specifications in the cooperative (heterogeneous) multi-agent scenario. A main challenge of the general strategy synthesis approach is the state-space explosion occurring during construction of a global model for agents with different, mutually dependent goals. Given a set of agents and their individual goal specifications represented through a local model and an LTL formula, we compute a compliant set of strategies that fulfill each agents' goal specification. We avoid the state-space explosion by computing individual solutions for each agent separately and then composing these solutions. During the initial strategy computation, assumptions over the states of other agents not represented in the local model are generated only where needed. These assumptions are resolved during composition of the individual solutions to assure compliance of the computed strategies. The effectiveness of this approach is demonstrated in several simulation case studies and compared to the classical, monolithic approach.**

## I. Introduction

Task assignment, task planning, and coordination of multiple autonomous, possibly heterogeneous agents is an active research field [1] with applications in multi-robot exploration [2] or automated warehouse solutions [3]. Tasks for a multi-agent team may vary from simple reachability to more sophisticated ones described in, for instance, PDDL or STRIPS.

In this work, we focus on tasks described in Linear Temporal Logic (LTL). The main benefit of LTL is its expressive power, allowing to define a wide variety of behaviours, constraints, and dependencies beyond the scope of traditional task, path and motion planning, and existence of a rich methodology for verification and strategy synthesis.

In this paper, strategy synthesis is understood as the automated process of generating behaviours of a system interacting with an environment from high-level specifications that are provably satisfying these specifications. Temporal logic strategy synthesis methods were applied to ensure safety and correctness of a range of control systems [4]; LTL has been used for path planning in hybrid systems [5], linear Systems [6] [7], as well as sampling-based methods [8].

In multi-agent systems, different approaches have been used to apply temporal logic strategy synthesis in a scalable fashion, avoiding the construction of a single monolithic synchronous product between the agents' individual models. Chen et al. [9] distribute the tasks from a fragment of LTL among the agents so that they can be executed independently. A receding horizon approach for deterministic systems is used to limit the planning scope [10], [11]. Guo et al. propose a leader-follower-scheme and switch leaders during execution to enforce satisfaction of all desired formulas [12]. Lin et al. synthesize reactive strategies with integrated motion planning for a fragment of LTL [13]. Lacerda et al. compose Petri nets representing multi-robot team restrictions [14]. Schillinger et al. work with probabilistic models of agents capturing uncertainties, and use multiple different methods in their works [15] [16], including efficient fragments of the LTL language, and reinforcement learning to optimize the agents' behaviour and keeping specification automaton and agent model separated.

None of the above works, however, handles full LTL specifications in reactive synthesis – a type of strategy synthesis that enables an agent to plan in reaction to various inputs from the environment [17]. Reactive synthesis comes at the cost of doubly exponential complexity, or lower expressivity of the task specification language, such as GR(1) [18] already in single-agent scenarios, and is especially challenging in multi-agent settings.

In this work, we aim to exploit dependencies and independencies between agents in a multi-agent scenario to tackle these scalability issues. We propose a solution where every agent first computes possible strategies in isolation, making different assumptions over the other agents only where it is necessary. Afterwards, the agents compose their plans and check if others can follow the made assumptions. To ensure completeness of our approach, we maintain the set of all candidate strategies in a suitable representation. During composition, agents' state spaces and candidate strategies are iteratively pruned, hindering the state-space explosion.

## II. Preliminaries

$\mathbb{N}$ is the set of natural numbers. $\varnothing$ is the empty set. We denote by $x(n)$ the n-th element of a tuple $x$. An *alphabet* $\Sigma$ is a finite set of symbols. A *word* $w \in \Sigma^\omega$ over an alphabet $\Sigma$ is an infinite sequence of these symbols and $\Sigma^\omega$ is the set of all infinite words. We denote by $w_i$ the i-th element of of a sequence $w$. $w_{i...}$ is the suffix of $w$, starting from the i-th element. The *complement* of a subset $X \subseteq \Sigma^\omega$ is defined as $\overline{X} = \Sigma^\omega \setminus X$.

**Definition II.1** (Atomic propositions)**.** Atomic propositions are propositions that are evaluated as true or false. A set of atomic propositions $AP$ is *consistent* if any atomic proposition and its negation do not belong to $AP$ at the same time.

*Division of Robotics, Perception and Learning, KTH Royal Institute of Technology Stockholm, Sweden schuppe@kth.se, tumova@kth.se

### A. Transition Systems

We model the possible behaviours of each agent as a transition system, following the usual definition (see e.g. [19]).

**Definition II.2.** (Transition System) A (non-deterministic) transition system is a tuple $T = (S, Act, \delta, s_{ini}, AP, L)$, where $S$ is a finite set of states, $Act$ is a finite set of inputs, $\delta : S \times Act \to 2^S$ is a transition function, $s_{ini}$ is the initial state, $AP$ is a finite set of atomic propositions and $L : S \to 2^{AP}$ is a state labeling function.

Without loss of generality, we assume that for each $a \in AP$, $\neg a$ also belongs to $AP$ and in each state $s \in S$, $L(s)$ contains either an atomic proposition or its negation.

**Definition II.3** (Path)**.** An (infinite) path $\pi$ on a transition system is a sequence of states $s_0 s_1 s_2 \dots$ such that $s_0 = s_{init}$ is the initial state and $\forall i \geq 0 : \exists a \in Act : s_{i+1} \in \delta(s_i, a)$.

**Definition II.4** (Trace)**.** A trace of a given path $\pi = s_0 s_1 s_2 \dots$ is the sequence of labels $\tau(\pi) = L(s_0)L(s_1)L(s_2)\dots$.

The traces of a transition system are thus (infinite) words on the alphabet $2^{AP}$, denoted by $Traces(T)$.

**Definition II.5** (Joint Trace)**.** Given $n$ transition systems $T_i = (S_i, Act_i, \delta_i, s_{0,i}, AP_i, L_i)$, $i \in [1, n]$, such that $AP_i \cap AP_j = \emptyset$, $\forall i \neq j$ and $n$ traces produced by these transition systems $\tau_i = o_{1,i} o_{2,i} \dots \in (2^{AP_i})^\omega$, we define the *joint trace* as a sequence:

$$\tau_{joint} = (o_{1,1} \cup o_{1,2} \cup \dots \cup o_{1,n})(o_{2,1} \cup o_{2,2} \cup \dots \cup o_{2,n}) \dots$$

**Definition II.6** (Control Strategy)**.** A (non-deterministic) *control strategy* $\lambda : S^* \to 2^{Act}$ maps a finite sequence of states to a subset of inputs of $T$. We define the traces of the transition system complying with the strategy as

$$Traces_\lambda(T) = \{\tau(\pi) \subseteq Traces(T) \mid \pi = s_0 s_1 s_2 \dots, \\ \forall i : a \in \lambda(s_0 \dots s_i) \text{ and } s_{i+1} \in \delta(s_i, a)\}.$$

### B. Linear Temporal Logic

We define our agents' specification, goal or restriction with a linear temporal logic (LTL) formula. The usual syntax and semantics are taken from [19].

**Definition II.7** (Syntax of LTL)**.** An LTL formula over a set of atomic propositions $AP$ is defined as follows:

$$\phi := true \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 U \phi_2 \mid \bigcirc \phi, \quad a \in AP$$

$\bigcirc\phi$ requires a proposition to hold in the *next* step. $\phi_1 U \phi_2$ requires $\phi_1$ to hold *until* $\phi_2$ holds. Using the operators $\neg$ and $\wedge$, the full power of propositional logic is obtained. Operators $\vee, \rightarrow, \leftrightarrow$ are defined as usual. Furthermore, $\Diamond\phi = true\, U \phi$ represents $\phi$ holding *eventually* in the future and $\Box\phi = \neg\Diamond\neg\phi$ *always* holding from now on. For full semantics, see [19].

### C. Automata on Infinite Words

Every LTL formula $\phi$ can be represented by an infinite word automaton $A_\phi$ [19].

**Definition II.8.** (Infinite Word Automata) An *infinite word automaton* over the alphabet $\Sigma$ is a tuple $A = (Q, \Sigma, \delta, q_{ini}, F)$, where $Q$ is a finite set of states, $\Sigma$ is a finite set of symbols, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $q_0$ is the initial state, and $F \subseteq Q$ is a finite set of *accepting* states.

**Definition II.9.** (Run) A *run* of the infinite word automaton $A$ on a word $\sigma = \sigma_0 \sigma_1 \sigma_2 \dots \in \Sigma^\omega$ is an infinite sequence of states $r = q_0 q_1 q_2 \dots \in Q^\omega$ with $q_{ini}$ being the initial state and $\forall i \geq 0 : q_{i+1} \in \delta(q_i, \sigma_i)$.

Given a word $\sigma$, we define the set of all possible runs of $A$ on $\sigma$ as $\text{Runs}_A(\sigma)$. We denote by $|r|_q$ the number of times the state $q$ appears in $r$.

**Definition II.10** (Acceptance conditions)**.** We consider three different *acceptance conditions*:

- non-deterministic Büchi (NBA): $\exists r \in \text{Runs}_A(\sigma) : \exists q \in F : |r|_q = \infty$
- universal co-Büchi (UCA): $\forall r \in \text{Runs}_A(\sigma) : \forall q \in F : |r|_q < \infty$
- universal k-co-Büchi (UKCA): $\forall r \in \text{Runs}_A(\sigma) : \sum_{q \in F} |r|_q \leq K$

We denote the sets of words accepted by those three acceptance conditions by $L_b(A), L_{uc}(A)$ and $L_{uc,K}(A)$. For every LTL formula $\phi$, there exists an Büchi automaton $A_\phi$ such that $L_b(A_\phi) = \text{Words}(\phi)$ [19]. By duality of accepting conditions, clearly $L_{uc}(A) = \overline{L_b(A)}$ holds [20]. From this, we can construct a UCA $A$ such that $L_{uc}(A) = \text{Words}(\phi)$ by constructing a (non-)deterministic Büchi automaton for $\neg\phi$ and changing the acceptance condition, leaving the automaton as is.

$$\text{Words}(\phi) = \{\sigma \mid \sigma \not\models \neg\phi\} = \overline{L_b(A_{\neg\phi})} = L_{uc}(A_{\neg\phi})$$

### D. Games

**Definition II.11** (Game)**.** A 2-player turn-based game is a tuple $G = (S_1, S_2, \Sigma_1, \Sigma_2, \Gamma_1, \Gamma_2, \delta_1, \delta_2, s_{ini})$, where

- $S_1$ and $S_2$ are the states of player 1 and 2, $S_1 \cap S_2 = \emptyset$;
- $\Sigma_1, \Sigma_2$ are finite sets of moves for player 1 and 2;
- $\Gamma_i : S_i \to 2^{\Sigma_i}$ are functions that assign to each state of player $i$ the moves that are available in that state;
- $\delta_i : S_i \times \Sigma_i \to S_j$ with $i, j = 1, 2$ and $i \neq j$ determines the state reached when player $i$ chooses a move from $\Gamma_i$ in a given state;
- $s_{ini} \in S_1$ is the initial state.

A play is a finite or an infinite alternating sequence of states $s_1 s_2 \dots$ with $s_1 = s_{ini}$, and $\forall i$, $\exists \gamma \in \Gamma_j(s_i)$ such that $\delta_j(s_i, \gamma) = s_{i+1}, j = 2 - (i \mod 2)$. If the play $s_1 s_2 \dots s_{n+i}$ is finite, we require $\Gamma_i(s_{n+i}) = \emptyset, i = 1, 2$. Each player plays according to a policy. For player $i$, a policy is a function $\lambda_i : (S_j S_l)^* \to \Sigma_i$, with $j = 2 - (i$

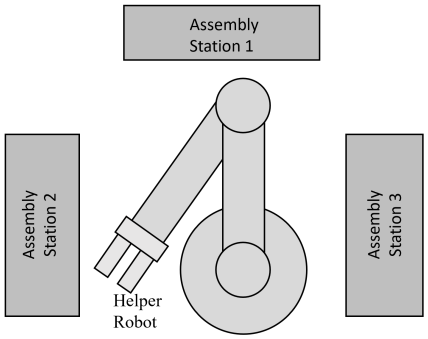


Fig. 1. Example from Sec. III-A. Multiple assembly stations share a helper robot.

$\mod 2), l = (2 - (i + 1) \mod 2)$. A policy induces a play as expected.

In a *safety game*, an objective is given through a set of states $Unsafe$. Player 1 wins if a play is infinite and does not intersect $Unsafe$. Acceptance conditions from Def. II.10 represent different winning objectives for infinite word automata that are interpreted as game; loosely speaking automata, whose states are partitioned between two players. In order for player 1 to win the game, a play needs to be an accepting run.

The set of all winning policies with respect to a safety objective is characterized by the *masterplan*, which is a history-independent policy $S_1 \to \Sigma_1$ for player 1 [21] and denoted by $MP$. Multiple algorithms to efficiently solve safety games exist. One such example is the OFTUR algorithm [22].

## III. Problem Definition

We propose a solution to the multi-agent task planning problem for a collaborative team of heterogeneous agents. The capabilities and possible behaviours of each agent will be modeled as a *non-deterministic transition system*, where the non-determinism models unknown outcomes of different actions. Each agent will need to satisfy an LTL formula representing task specification and constraints. The formula of each agent might be dependent on other agents.

**Problem III.1.** Given $n$ transition systems $T_i = (S_i, Act_i, \delta_i, s_{ini,i}, AP_i, L_i)$, $i \in [1, n]$, such that $AP_i \cap AP_j = \emptyset$, $\forall i \neq j$ and LTL formulas $\phi_1, \ldots, \phi_n$ defined over $AP = \bigcup_{i=1}^{n} AP_i$ each, find control strategies $\lambda_i : (S_1 \times \cdots \times S_n)^* \subseteq Act_i$ such that resulting joint traces complying with these strategies satisfy all $\phi_i$:

$$\tau_{joint} \models \phi_1 \wedge \cdots \wedge \phi_n$$

### A. Example

Consider an industrial assembly line with assembly robots situated at different stations. Each robot performs a sequence of preparation and assembly steps to fabricate a workpiece. Some of these steps may be difficult and require assistance. For this reason, additional robot arms are installed in reach

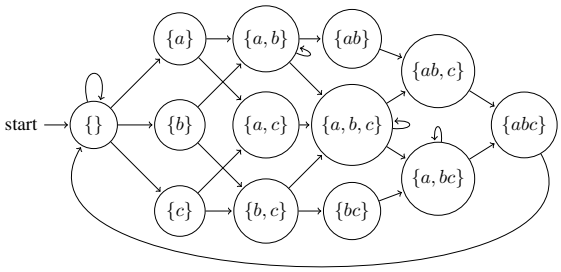


Fig. 2. Transition system of a single assembly station. Action labels and state names are omitted due to readability.

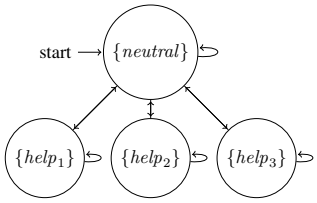


Fig. 3. Transition system of a helper robot for three assembly stations. Action labels and state names are omitted due to readability.

of the assembly stations. Since the assembly stations share a resource, coordination of assembly strategies is needed to avoid conflicts and minimize waiting times. An example of a setup with three assembly stations and one helper robot is illustrated in Figure 1. An example of a transition system modeling an assembly robot is given in Figure 2, where $a$, $b$, $c$ stand for three different types of pieces that need to be prepared for assembly. This could represent polishing or applying adhesives. $ab$ and $bc$ stand for intermediate workpieces and $abc$ is the finished workpiece. The transitions correspond to the robot's capabilities to either prepare a part or do a specific assembly. The self-loops allow the robot to wait for help, which the assembly of $a$ and $b$ to $ab$ requires. This constraint is stated in a constraint formula, which can be added to the goal specification through a conjunction. The helper can be modeled as the transition system in Figure 3. It can provide assistance only to a specific assembly station at a time. The individual tasks are given in (1) and (2). The formula for assembly station $i$ consists of the goal specification and model constraints: keep assembling the final workpiece $abc_i$, and assemble the connection between piece $a$ and $b$ if assistance from the helper robot is provided. The helper task is to repeatedly return to the neutral position, to allow the robot arm to provide assistance where needed.

$$\phi_{assembly,i} = \square\lozenge abc_i \wedge \square(a_i \wedge \bigcirc(ab_i \vee abc_i) \to help_i) \quad (1)$$

$$\phi_{helper} = \square\lozenge neutral \quad (2)$$

**Remark III.1** (Monolithic Approach)**.** A straightforward approach to solving problems with multiple models and multiple specifications by composing multiple transition systems $T_i, i \in [1, n]$ into a single, global transition system:

$$T = \bigotimes_{i \in [1,n]} T_i$$

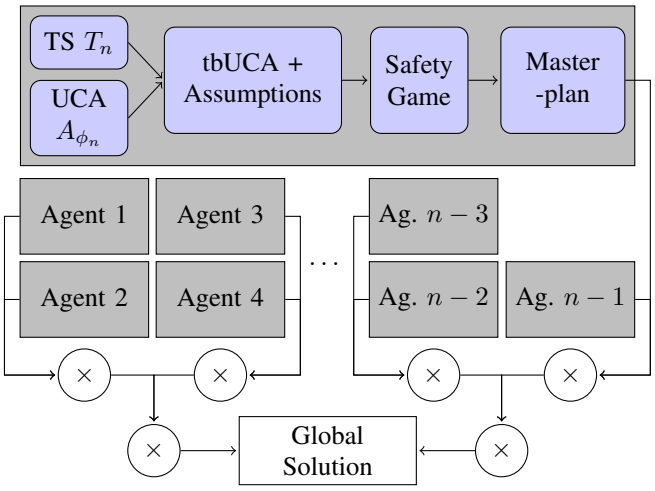


Fig. 4. Construction of the individual solutions and their binary composition for multiple agents

with composition defined by the synchronous product operator [19]. Fulfillment of all specification formulae can be assured by conjunction of all partial specifications:

$$\phi = \bigwedge_{i \in [1,n]} \phi_i$$

From here, our defined problem can be solved by either constructing an automaton similar to our method and using methods from [20] or constructing $T \otimes A_\phi$ and applying methods from [4]. While those methods are correct and complete in theory, scalability is an issue that cannot be overcome by adding computational power [23]. For this reason, we introduce a compositional method inspired by [21] to overcome these scalability issues.

## IV. Compositional Multi-Agent Strategy Synthesis

### A. Method Outline

A flowchart illustrating the proposed compositional process addressing Problem III.1 is shown in Figure 4. As a starting point, we translate each specification formula $\phi_i$ to an UCA $A_{\phi_i}$ as shown in Sec. II-C. For each pair $T_i$ and $A_{\phi_i}$, we define a product *turn-based UCA (tbUCA)* $P_i$ that captures the possible trajectories of $T_i$ accepted by the specification automaton $A_{\phi_i}$, as well as the dependencies on other agents via additional labels on the transitions of the tbUCA, which we call *assumptions*. We apply methods from [20], to effectively reduce the tbUCA with assumptions to a *safety game*, for which efficient solution algorithms exist [22]. We obtain a master plan as an efficient representation of all candidate strategies.

To find strategies that are mutually compatible with the assumptions of the tbUCAs $P_1, \dots P_n$, we pairwise incrementally compose the corresponding master plans. This way, the candidate strategies that are not mutually compliant between agents are gradually pruned away. Finally, the resulting winning region is a representation of all possible strategies satisfying Problem III.1.

### B. Candidate Strategies for a Single Agent

*1) Product Automaton with Assumptions:* We extend the usual definition of a synchronous product between a transition system and an infinite word automaton [4], by introducing additional labels to the transition function $\delta$, called assumptions. These labels are subsets of atomic propositions of the other agents that need to hold true for the transition to happen. Thus, we represent dependencies between agents without explicitly modelling possible states of every other agent, which would have lead to a state-space explosion.

**Definition IV.1** (Product tbUCA with Assumptions)**.** Given a transition system $T = (S, Act, \delta_{ts}, s_{ini}, AP_i, L)$ and a UCA $A_\phi = (U, 2^{AP_\phi}, \delta_{uc}, u_{ini}, F)$, we define the assumption alphabet $\widehat{\Sigma} = 2^{\widehat{AP}}$, where $\widehat{AP} = AP_\phi \backslash AP_i$. $\sigma \in \widehat{\Sigma}$ is called an assumption. *Product turn-based UCA (tbUCA) with assumptions* is a tuple $P = (Q_1, Q_2, \Sigma_1, \Sigma_2, \delta_1, \delta_2, q_{ini}, Q_F)$, where

- $Q_1 \subseteq S \times U$; $Q_2 \subseteq S \times U \times Act \times \widehat{\Sigma}$,
- $\Sigma_1 = Act \times \widehat{\Sigma}$; $\Sigma_2 = S$,
- $\delta_1 : Q_1 \times \Sigma_1 \to Q_2$. $\delta_1((s,u),(a,\sigma)) = \{(s,u,a,\sigma) \in Q_2 \mid \delta_{ts}(s,a) \neq \varnothing\}$,
- $\delta_2 : Q_2 \times \Sigma_2 \to Q_1$. $\delta_2((s,u,a,\sigma),s_i) = \{(s_i,u_i) \in Q_1 \mid u_i \in \delta_{uc}(u,\widehat{\sigma})\}$ , if $s_i \in \delta_{ts}(s,a)$ and $\widehat{\sigma}$ is the largest element of $2^{AP_\phi}$ that is also a subset of $L(s_i) \cup \sigma$,
- $q_{ini} = (s_{ini}, u_{ini}) \in Q_1$ is the initial state and
- $Q_F = \{(s,u) \mid (s,u) \in Q_1 \text{ and } u \in F\}$.

States in $Q_1$ represent the current state of the transition system $s$ and the UCA $u$. The deterministic transitions from $Q_1$ to $Q_2$ represent the agent choosing an action $a \in Act$ available in the current state of $T$ and making an assumption $\sigma \in \widehat{\Sigma}$ about the other agents' atomic propositions. This assumption may be seen as a requirement for the agents' cooperation. States in $Q_2$ are intermediate states including the chosen action and assumption on top of $s$ and $u$. The transitions from $Q_2$ to $Q_1$ respect the transitions in $A_\phi$ and are labeled with the non-deterministic outcomes of corresponding transitions in $T$.

**Definition IV.2** (Run on $P$)**.** A *run* of $P$ on a word $w = w_1 w_2 \cdots \in (\Sigma_1 \Sigma_2)^\omega$ is a alternating sequence of states $q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$ such that $q_1$ is the initial state and $\forall i \geq 1 : q_{i+1} \in \delta_j(q_i, w_i)$ with $j = 2 - (i \mod 2)$).

**Definition IV.3** (Traces of $P$)**.** The *trace* $\tau \in (2^{AP})^\omega$ of a run $r = q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$ of the tbUCA $P$ over the word $w \in (\Sigma_1 \Sigma_2)^\omega$ is defined as:

$$\tau_i(r) = L(q_{2i}(1)) \cup q_{2i}(4), \forall i$$

The trace of a product tbUCA with assumptions is a sequence of the propositions holding in the current state of $T$ and the propositions of other agents which are assumed to hold. We denote by $Traces(P)$ all traces of infinite runs on $P$ and by $Traces_{acc}(P)$ all possible traces of words accepted by $P$. An important property of the traces of $P$

is their correspondence to traces of the agents and words accepted by $A_\phi$ :

**Lemma IV.1** (Accepted traces of $P$)**.** Let $P$ be a product tbUCA with assumptions of $T$ and $A_\phi$.

$$\begin{aligned}\textit{Traces}_{acc}(P) =&\{w \in (2^{AP})^\omega \mid w \in L_{uc}(A_\phi) \text{ and}\\ &\exists \tau \in \textit{Traces}(T), \text{ such that } \forall j.\tau_j \subseteq w_j\}.\end{aligned}$$

*Proof:* "$\subseteq$": For each run $r = q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$ on $P$, it holds that $q_{2i-1}(1) = q_{2i}(1), q_{2i-1}(1) \in S, q_{2i+1}(1) \in \delta_{ts}(q_{2i}(1), q_{2i}(3)), \forall i \geq 1$ and $q_1(1)$ is the initial state of $T$. So the sequence $q_1(1)q_3(1)q_5(1), \cdots \in S^\omega$ is a path on $T$. Following from this and Def. IV.3,

$$\begin{aligned}\textit{Traces}(P(T,\phi)) \subseteq&\{w \in (2^{AP})^\omega \mid \exists \tau \in \textit{Traces}(T),\\ &\text{such that } \forall j.\tau_j \subseteq w_j\}\end{aligned}$$

Also, $q_{2i-1}(2) = q_{2i}(2), q_{2i-1}(2) \in Q, q_{2i+1}(2) \in \delta_{uc}(q_{2i}(2), L(q_{2i}(1)) \cup q_{2i}(4)), \forall i \geq 1$ and $q_1(2)$ is the initial state of $A$. So the sequence $q_1(2)q_3(2)q_5(2), \cdots \in Q^\omega$ is a run on $A$. A run on $P$ is accepting, iff the corresponding run on $A$ is accepting. Every run on $P$ that visits a final state $q_f \in Q_F$ infinitely often is rejected by $P$. This run $r$ of $P$ has has a corresponding run on $A$, which visits a final state in $F$ infinitely often, since $q_F(2) \in F$. So the run is not accepting on A as well. From this follows $\text{Traces}_{acc}(P(T,\phi)) \subseteq L_{uc}(A)$. Together with the above result, it holds that

$$\begin{aligned}\textit{Traces}_{acc}(P) \subseteq&\{w \in (2^{AP})^\omega \mid w \in L_{uc}(A_\phi) \text{ and}\\ &\exists \tau \in \textit{Traces}(T), \text{ such that } \forall j.\tau_j \subseteq w_j\}.\end{aligned}$$

"$\supseteq$": We show that for every infinite path $t = s_1 s_2 \cdots \in S^\omega$ on $T$ and every word $w \in \Sigma_\phi = 2^{AP_\phi}$ with $w \models \phi$ and $w_i \subseteq L(t_i), \forall i$, there exists a run $r = q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$ on $P$. By Def. II.10, all runs $u = u_1 u_2 \cdots \in Q^\omega$ on $A$ over $w$ are accepting. We will now construct this accepting run $r = q_1 q_2 \cdots \in (Q_1 Q_2)^\omega$ on $P$. $q_1 = (s_1, u_1)$ is the initial state. The states of the runs are defined as $q_2 i - 1 = (s_i, u_i) \in Q_1, \forall i \geq 1$ and $q_2 i = (s_i, u_i, a, \sigma) \in Q_2$ with $a \in \textit{Act}, t_{i+1} \in \delta_{ts}(s_i, a)$ and $\sigma \in \widehat{AP}$ with $u_{i+1} \in \delta_{uc}(u_i, \sigma \cup L(s_{i+1})), \forall i \geq 1$. The run $r$ is accepting since $w \models phi$ and so every run $u$ is accepting. Since for every path on $T$ and a word $w$ we constructed an accepting run on $P$, it holds that

$$\begin{aligned}\textit{Traces}_{acc}(P) \supseteq&\{w \in 2^{AP} \mid w \in L_{uc}(A_\phi) \text{ and}\\ &\exists \tau \in \textit{Traces}(T), \text{ such that } \forall j.\tau_j \subseteq w_j\}.\end{aligned}$$

□

A product tbUCA with assumptions can be interpreted as a 2-player turn-based game as in Def. II.11. $Q_1$ are states of player 1 and $Q_2$ the states of player 2. Each run on $P$ is a play. A tbUCA is *realizable* if there exists a winning policy for player 1 such that for all policies of player 2, the resulting play is an accepting run [17].

With respect to Problem III.1, realizability of all $P_i, i \in [1, n]$ is necessary for the existence of a set of strategies $\lambda_i$ fulfilling all $\phi_i$, but not sufficient, as made assumptions in each $P_i$ need to be checked for compatibility with each $P_j, j \neq i$.

*2) From tbUCA to tbUKCA to Safety Games:* Filiot et al. [20] showed that it is possible to reduce the realizability problem of a tbUCA automaton to the realizability of a tbUKCA automaton, a *turn-based universal K-co-Büchi Automaton*[1], by finding a finite boundary for $K$.

Furthermore, they determinize the automaton to $det(P, K)$. Each state $F \in \mathcal{F}_1$ ($\mathcal{F}_2$) in $det(P, K)$ is a mapping from all $q \in Q_1$ ($q \in Q_2$) to $[0, K+1]$ and called a *counting function*. In each state of $det(P, K)$, the counter of a state $q \in Q_1 \cup Q_2$ indicates the maximum number of times an accepting state has been visited over all possible runs on the prefix read that ends up in $q$. If no run on the prefix read ends in $q$, then the counter is set to -1. This way, the deterministic tbUKCA becomes a safety game $G(P, K) = (\mathcal{F}_1, \mathcal{F}_2, \Sigma_1, \Sigma_2, \Gamma_1, \Gamma_2, \Delta_1, \Delta_2, F_{ini})$ with appropriately defined $\Delta_1$ and $\Delta_2$, and $\Gamma_1, \Gamma_2$ allowing all possible $\sigma \in \Sigma_1, \Sigma_2$ in all states. We aim to avoid a visit to $\textit{Unsafe} = \mathcal{F}_f = \{F \in \mathcal{F}_1 \cup \mathcal{F}_2 \mid \exists q, F(q) > K\}$. The complete definition is given in [20].

**Lemma IV.2** ([20])**.** Let $P$ be a tbUCA over inputs $\Sigma_2$ and outputs $\Sigma_1$ with $n$ states and $K = 2n(n^{2n+2} + 1)$. Then $P$ is realizable iff player 1 has a winning policy in the safety game $G(P, K)$.

Guided by Lemma IV.2, we reduce the realizability problem for $P$ to finding a masterplan $MP(G(P, K))$ in a safety game with assumptions.

### C. Composition of Candidate Strategies

For every agent $T_i$ and the corresponding specification $\phi_i$, we created a tbUCA with assumptions $P_i$ and transformed into a safety game $G(P_i, K)$. We also have a masterplan $MP$ for each $G(P_i, K)$ representing all candidate strategies. The remaining step is the composition of the masterplans, which is inspired by [21], but requires additional reasoning over the formerly made assumptions. We define composition through the product of two safety games with assumptions that "resolves" the mutual assumptions the two games impose on each other, while keeping the remaining ones.

First, let us introduce an observation and necessary definitions allowing us to map the composition of the masterplans of the safety game onto the composition of the underlying transition systems. Let $P$ be a tbUCA with assumptions as defined in Def. IV.1, $G(P, K)$ a safety game constructed from $P$ as outlined above. For every state $F \in \mathcal{F}_1$ it holds that $s_i = s_j$, for every $q_i, q_j \in Q_1$ where $F(q) \neq -1$. For every state $F \in \mathcal{F}_2$ it holds that $s_i = s_j$, $a_i = a_j$ and $\sigma_i = \sigma_j$ for every $q_i, q_j \in Q_2$ where $F(q) \neq -1$.

We denote the projection of states in $\mathcal{F}_i$ onto the states of $T$ and $A_\phi$ as $\mathit{States}_{TS} : \mathcal{F}_i \to S$ and $\mathit{States}_{UCA} : \mathcal{F}_i \to 2^U$, where $i \in \{1, 2\}$. Note that thanks to the observation above, $\mathit{States}_{TS}(\mathcal{F})$ is a singleton for all $\mathcal{F}$. The labeling function $L$ can be naturally extended to $G(P, K)$ by inheriting the labels from the respective states of $T$. We also define $\mathit{Assum} : \mathcal{F}_2 \to 2^{\widehat{AP}}$ as the assumed proposition.

[1]A tbUKCA is a turn-based universal K-co-Büchi Automaton with the accepting condition threshold defined by $K$ as in Def. II.10.

**Definition IV.4.** (Product of Safety Games with Assumptions) Let $G^i = (\mathcal{F}_1^i, \mathcal{F}_2^i, \Sigma_1^i, \Sigma_2^i, \Gamma_1^i, \Gamma_2^i, \Delta_1^i, \Delta_2^i)$ with propositions $AP^i$, assumption symbols $\widehat{AP}^i$, $i \in \{1,2\}$, be two safety games with assumptions. The product is defined as a safety game with assumptions $G^1 \otimes_a G^2$:

- $\mathcal{F}_i^p \subseteq \mathcal{F}_i^1 \times \mathcal{F}_i^2,\ i \in \{1,2\}$,
- $\Sigma_i^p = \Sigma_i^1 \times \Sigma_i^2; i \in \{1,2\}$,
- $\Delta_1^p : \mathcal{F}_1^p \times \Sigma_1^p \to \mathcal{F}_2^p$,
- $\Delta_2^p : \mathcal{F}_2^p \times \Sigma_2^p \to \mathcal{F}_1^p$,
- $\Gamma_i^p : \mathcal{F}_1^p \to 2^{\Sigma_i^p}$ with $\Gamma_1^p((F_i^1, F_i^2)) = \{(\gamma^1, \gamma^2) \mid \gamma^1 \in \Gamma_i^1(F_i^1), \gamma^2 \in \Gamma_i^2(F_i^2)\},\ i \in \{1,2\}$,
- $AP^p = (AP^1 \cup AP^2)$ is the new set of propositions,
- $AP_r^p = (\widehat{AP}^1 \cup \widehat{AP}^2) \setminus AP^p$ are the propositions that will be "resolved",
- $\widehat{AP}^p = (\widehat{AP}^1 \cup \widehat{AP}^2) \setminus AP_r^p$ is the new set of remaining assumption symbols,
- $States_{TS}^p(F) = States_{TS}^1(F) \cup States_{TS}^2(F)$,
- $States_{UCA}^p(F) = States_{UCA}^1(F) \cup States_{UCA}^2(F)$,
- $Assum^p(F) = (Assum^1(F) \cup Assum^2(F)) \setminus AP_r$,

with $\Delta_1^p((F_1^1, F_1^2), (\gamma^1, \gamma^2)) =$

$$\{(F_2^1, F_2^2) \mid F_2^1 = \Delta_1^1((F_1^1, \gamma^1), F_1^2 = \Delta_1^2(F_1^2, \gamma^2) \text{ and } Assum^1(F_2^1) \cup Assum^2(F_2^2) \text{ is consistent.}\}$$

and $\Delta_2^p((F_2^1, F_2^2), (\gamma^1, \gamma^2)) =$

$$\{(F_2^1, F_2^2) \mid F_2^1 = \Delta_1^1((F_1^1, \gamma^1), F_1^2 = \Delta_1^2(F_1^2, \gamma^2) \text{ and } Assum^1(F_2^1) \cup Assum^2(F_2^2) \cup L^p((F_2^1, F_2^2)) \text{ is consistent.}\}$$

Any play on the game $G^p = G^1 \otimes_a G^2$ is a play both on $G^1$ and $G^2$. Any play complying with $MP(G^p)$ is winning for player 1 and the corresponding plays on $G^1$ and $G^2$ are also winning for player 1. Furthermore, for each pair of plays on $G^1$ and $G^2$, there exists a corresponding play on $G^p$. If both plays on $G^1$ and $G^2$ are complying with their respective masterplans, they are both winning for player 1 in $G^1$ and $G^2$. If at every state, the assumptions of the plays are consistent (see Def. II.10) with each other and with both labelling functions of their respective encoded transition systems, the constructed play on $G^p$ is also winning for player 1 and thus a part of the masterplan of $G^p$.

*1) Product pruning:* For the sake of computational efficiency, it is important to note that the search of a master plan can be restricted to the reachable part of the safety game; thus, pruning away potentially large unreachable parts of the product throughout the compositions. Formally, we denote by $Reach(G, F_{ini})$ the set of states reachable from $F_{ini}$ in $G$.

**Definition IV.5** (Masterplan of Reachable Positions [21])**.** The *master plan of reachable positions* is defined as

$$MP_R(G, F_{ini})(F) = \begin{cases} MP(G)(F), \text{if } F \in Reach(G, F_{ini}) \\ \varnothing, \text{ else} \end{cases}$$

**Definition IV.6** (Restricted Games)**.** Given a function $\Lambda : S_1 \to 2^{\Sigma_1}$, in the *restricted game* $G[\Lambda]$, the available moves $\Gamma_1$ are further restricted by $\Lambda$, i.e. $G[\Lambda] = (\mathcal{F}_1, \mathcal{F}_2, \Sigma_1, \Sigma_2, \Gamma_1{}', \Gamma_2, \Delta_1, \Delta_2)$ with $\Gamma_1{}'(f) = \Gamma_1(f) \cap \Lambda(f)$.

The following lemma guides us in efficiently composing safety games with assumptions and is a slightly modified version of a lemma found in [21].

**Lemma IV.3.** Let $\Lambda_1 = MP(G^1)$ and $\Lambda_2 = MP(G^2)$.

$$MP(G^1 \otimes_a G^2) = MP(G^1[\Lambda_1] \otimes_a G^2[\Lambda_2]).$$

*2) Incremental Composition:* In practice, to efficiently find strategies to Problem III.1, we use the following property of tbUKCA [20]:

$$\forall k_1, k_2, 0 \le k_1 \le k_2 : L_{uc,k_1}(P) \subseteq L_{uc,k_2}(P) \subseteq L_{uc}(P).$$

Intuitively, this property allows us to find the masterplans in a incremental fashion, starting with $k = 0$ and iteratively raising k until compatible masterplans in $G(P, k)$ are calculated through composition or the theoretical boundary of $2n(n^{2n+2} + 1)$ is reached. Methods to construct strategies from the composed masterplans are known [20]. This incremental idea finally leads to Alg. 1.

**Algorithm 1:** Compositional Synthesis

**Data:** $T_1, \ldots, T_n$ and $\phi_1, \ldots, \phi_n$

$\mathcal{P} = \{P_i(T_i, A_{\phi_i}), \mid i \in [1,n]\}$ ; // tbUCA with asm.

$s = \max\{\ |P| \mid P \in \mathcal{P}\}$;

$thresh = 2s(s^{2s+2} + 1)$ ; // upper bound for k

**for** $k = 0$ **to** *thresh* **do**

  $\mathcal{P}_{det} = \{det(P, k) \mid P \in \mathcal{P}\}$ ; // determinize $P_i$

  /* calculate masterplan of reachable positions for all ind. games */

  $\mathbb{G} = \{G^i[\Lambda_i] \mid \Lambda_i = MP_R(G(P_i, k), F_{ini})\}$

  /* compose games until global game is reached */

  **while** $|\mathbb{G}| > 1$ **do**

    **if** $\exists i : G^i \in G = \emptyset$ **then**

      break ; // one of the games has no solution, retry with higher k

    $\mathbb{G}_{new} = \emptyset$;

    find pairing of games in $\mathbb{G}$;

    **for** *each pair found above* $(G^i, G^j)$ **do**

      $G_{prod} = G^i \otimes_a G^j$;

      $\Lambda_{prod} = MP_R(G_{prod}, F_{ini})$;

      $\mathbb{G}_{new} = \mathbb{G}_{new} \cup \{G_{prod}[\Lambda_{prod}]\}$;

    $\mathbb{G} = \mathbb{G}_{new}$ ; // during each iteration, the total number of games halves

  **return** $MP_R(\mathbb{G}[0])$ ; // all games are composed into one global game

### D. Analysis and Discussion

**Theorem IV.1.** Alg. 1 is a correct and complete solution to Problem III.1.

*Proof:* Let $T_i$ and $\phi_i, i \in [1,n]$ be $n$ agent models and specifications. Let $\Lambda_{res}$ be a non-empty masterplan of the final composed Game $G$ in Alg. 1 for some $K \in \mathbb{N}$. From Lemma IV.1 follows that realizability of all $P_i(T_i, A_{\phi_i})$ is a

necessary condition for strategies $\lambda_i$ as defined in Sec. III to exist. Furthermore, if $\lambda_i$ exist, they have to be a subset of the strategies realizing $P_i(T_i, A_{\phi_i})$. Lemma IV.2 shows that individual masterplans of $G_i, i \in [1, n]$ are representing all policies realizing $P_i(T_i, A_{\phi_i})$. Through repeated application of Lemma IV.3, the allowed moves in $\Lambda_{res}$ encode allowed moves in each individual subgame $G^i$ and their respective master plans $\Lambda_i$. If all games $G^i$ are composed, $\widehat{AP} = \varnothing$. If $\Lambda_{res}$ is non-empty, all outcomes of a play with $\Lambda_{res}$ are both winning and complying with all $T_i$ by Def. IV.4. So $\Lambda_{res}$ is indeed encoding all possible policies $\lambda_i$ as defined in Sec. III. To conclude that there exist no strategies $\lambda_i$, the upper boundary $thresh$ in Alg. 1 needs to be reached. □

## V. Experimental Results

We implemented the algorithm in Python3. Computation has been performed on a laptop with Intel i5-7300 processor and 8GB RAM.

*1) Comparison to the monolithic approach:* First, we revisit the running example in Sec. III-A and compare our method to the monolithic approach. The results for varying numbers of assembly stations are given in Table I. The 'run' shows that the compositional method is able to handle larger instances and is able to calculate solutions faster than the monolithic approach. The main factor in scalability is the amount of pruning due to the use of restricted games in the composition. As all assembly stations are identical, there exist multiple equivalent strategies that are all compatible during each composition. This leads to a large growth of possible strategies (and therefore considered states) during each composition step. With three assembly stations, the helper arm is used continuously (see Table II). The introduction of a fourth assembly leads to a large number of waiting cycles for each assembly station which leads to a sharp increase in considered states.

*2) Heterogenous Capabilities:* We further explore a scenario where assembly stations are not identical, as the homogeneity of models is not necessary in our approach. One assembly station is now disallowed to have more than two individual pieces prepared at the same time, while the other two can only assemble $ab$ before $bc$ and vice versa. Heterogeneity of agents does not impose additional complexity to the computation; in contrast, synthesizing a strategy is slightly more efficient in comparison the homogeneous scenario, as the added restrictions further limit the number of possible strategies during composition (see case 'het' in Table I).

*3) Reactivity:* Finally, we consider a scenario where the assembly stations get a simpler assembly task of a two-component workpiece $ab$, but are unable to choose to pick up a specific component and need to react according to the component they get. This non-deterministic scenario was also handled by our approach successfully and is reported on in in 'react' in Table I).

*4) Scalability:* While our method could outperform the monolithic approach in Sec. V-.1, it also suffered from the state explosion problem with more than three assembly stations. This is due to a high number of candidate strategies in each step of the composition process. We consider yet another scenario with a set of assembly robots organized in a ring. Each robot can either prepare their own assembly or help a neighbouring robot with theirs. A robot that has started an assembly cannot finish without assistance from both neighbouring robots, so a team strategy is needed to prevent deadlocks. The task is formalized through the following LTL formula.

$$\phi_i = \square\lozenge n_i \wedge \square\lozenge a_i \wedge \square(a_i \wedge \bigcirc\neg a_i \rightarrow h_{(i-1)\%n} \wedge h_{(i+1)\%n}).$$

Our approach can handle over 500 robots, while the monolithic approach already struggles with 4 robots (see Table III). Here, a high degree of dependency between the robots led to restrictive assumptions, low number of candidate strategies, and extensive pruning in intermediate products.

### A. Discussion

*1) Composition Order:* Pruning of the intermediate products is a critical factor in scalability of our approach. We thus propose to compose those intermediate products in Alg. 1 that present the largest number of "resolved" assumptions for each pair and thus reducing the assumption alphabet as quickly as possible.

*2) Optimality:* Our approach does not guarantee optimality of the resulting plans. However, since we are iteratively raising $K$ in our incremental composition (see Sec. IV-C.2) , we will first find candidate strategies with the lowest number of steps between of consecutive visits to states that mark a progress towards fulfillment of liveness specifications[2].

*3) Long-Term Autonomy and Replanning:* Our approach can be used to efficiently replan in case of a change in the environment, as only one half of the composition tree needs to be redone. One could define a heuristic which defines the composition order, to compose the robots with the lowest probability of changing during long-term execution first.

## VI. Conclusion and Further Directions

We have proposed a novel, compositional approach to the problem of task planning for cooperative, possibly heterogeneous multi-agent systems under LTL specifications. Future work will include more compact representation of assumptions and partial solutions. Extending the agent models to include costs for actions will enable the method to optimize certain variables, as time or energy consumption. Expanding the notion of assumptions to the (semi-)competitive scenario could lead to interesting applications in planning with humans in the loop.

## ACKNOWLEDGMENT

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation, the EU H2020 Research and Innovation Programme under GANo. 731869 (Co4Robots) and the Swedish Research Council (VR).

[2]Imagine $\phi = \square\lozenge x$. This formula holds if $\neg x$ does not hold in every step at some point in the trajectory. It would hold if $x$ holds every third state, but it also would hold if $x$ holds every 100th state.

| | | Monolithic | | | Compositional | | |
|---|---|---|---|---|---|---|---|
| case | assembly stations | tbUCA size | states considered | time | max tbUCA size | max states cons. | time |
| run | 1 | 413 (806) | 2036 | 2.1sec | 131 (225) | 449 | 1.0 sec |
| | 2 | 20998 (50700) | - | - | 131 (225) | 6008 | 3.1 sec |
| | 3 | 872983 (2353750) | - | - | 131 (225) | 9589 | 3.2 sec |
| | 4 | - | - | - | 131 (225) | - | - |
| het | 3 | 447776 (1191218) | - | - | 112 (196) | 2449 | 0.8 sec |
| react | 3 | 19440 (63903) | - | - | 40 (68) | 9923 | 2.6 sec |

TABLE I

SIZE OF SOLUTIONS AND CONSIDERED STATES IN THE MONOLITHICAL AND COMPOSITIONAL METHOD. TBUCA SIZE IS GIVEN AS VERTICES (EDGES).

| assembly1 | assembly2 | assembly3 | helper |
|---|---|---|---|
| _ | _ | ... | |
| _ | $a$ | $a, c$ | $neutral$ |
| _ | $a, b$ | $a, b, c$ | $help_3$ |
| $c$ | $a, b, c$ | $ab, c$ | $neutral$ |
| $b, c$ | $a, b, c$ | $abc$ | $help_2$ |
| $bc$ | $ab, c$ | _ | $neutral$ |
| $a, bc$ | $abc$ | $c$ | $help_1$ |
| $abc$ | _ | $b, c$ | $neutral$ |
| _ | $a$ | $ab, c$ | $help_3$ |
| $b$ | $a, b$ | $abc$ | $neutral$ |
| $a, b$ | $a, b, c$ | $abc$ | $help_2$ |
| $a, b, c$ | $ab, c$ | _ | $neutral$ |

TABLE II

EXECUTION OF A SYNTHESIZED STRATEGY WITH DIFFERENT STARTING POINTS. COLOURED CELLS INDICATE USE OF THE HELPER.

| stations | max tbUCA size | max states cons. | time |
|---|---|---|---|
| 2 | 42 (80) | 79 | 0.1 sec |
| 4 | 42 (80) | 88 | 0.25 sec |
| 10 | 42 (80) | 88 | 0.69 sec |
| 20 | 42 (80) | 88 | 1.43 sec |
| 40 | 42 (80) | 88 | 2.84 sec |
| 100 | 42 (80) | 88 | 8.66 sec |
| 200 | 42 (80) | 88 | 16.28 sec |
| 500 | 42 (80) | 88 | 42.57 sec |

TABLE III

"DINING PHILOSOPHERS".

## REFERENCES


[1] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[2] L. Matignon, L. Jeanpierre, and A.-I. Mouaddib, "Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes," in *Twenty-sixth AAAI conference on artificial intelligence*, 2012.

[3] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2016, pp. 1144–1152.

[4] C. Belta, B. Yordanov, and E. A. Gol, *Formal methods for discrete-time dynamical systems*. Springer, 2017, vol. 89.

[5] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 4885–4890.

[6] P. Tabuada and G. J. Pappas, "Model checking LTL over controllable linear systems is decidable," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2003, pp. 498–513.

[7] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[8] Y. Kantaros and M. M. Zavlanos, "Distributed optimal control synthesis for multi-robot systems under global temporal tasks," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 162–173.

[9] Y. Chen, X. C. Ding, and C. Belta, "Synthesis of distributed control and communication schemes from global LTL specifications," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE, 2011, pp. 2718–2723.

[10] J. Tumova and D. V. Dimarogonas, "A receding horizon approach to multi-agent planning from local LTL specifications," in *2014 American Control Conference*. IEEE, 2014, pp. 1775–1780.

[11] ——, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239–248, 2016.

[12] M. Guo, J. Tumova, and D. V. Dimarogonas, "Cooperative decentralized multi-agent control under local LTL tasks and connectivity constraints," in *53rd IEEE conference on decision and control*. IEEE, 2014, pp. 75–80.

[13] A. Partovi, R. R. da Silva, and H. Lin, "Reactive integrated mission and motion planning for mobile robotic manipulators," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 3538–3543.

[14] B. Lacerda and P. U. Lima, "Petri net based multi-robot task coordination from temporal logic specifications," *Robotics and Autonomous Systems*, vol. 122, p. 103289, 2019.

[15] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Decomposition of finite LTL specifications for efficient multi-agent planning," in *Distributed Autonomous Robotic Systems*. Springer, 2018, pp. 253–267.

[16] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Hierarchical LTL-task mdps for multi-agent coordination through auctioning and learning," *The international journal of robotics research*, 2019.

[17] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1989, pp. 179–190.

[18] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive (1) designs," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2006, pp. 364–380.

[19] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[20] E. Filiot, N. Jin, and J.-F. Raskin, "Antichains and compositional algorithms for LTL synthesis," *Formal Methods in System Design*, vol. 39, no. 3, pp. 261–296, 2011.

[21] ——, "Compositional algorithms for LTL synthesis," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2010, pp. 112–127.

[22] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Efficient on-the-fly algorithms for the analysis of timed games," in *International Conference on Concurrency Theory*. Springer, 2005, pp. 66–80.

[23] C. S. Althoff, W. Thomas, and N. Wallmeier, "Observations on determinization of büchi automata," in *International Conference on Implementation and Application of Automata*. Springer, 2005, pp. 262–272.