

# Cooperative Tasking for Deterministic Specification Automata

Mohammad Karimadini, Hai Lin, Ali Karimoddini

## ABSTRACT

This paper proposes necessary and sufficient conditions for task decomposability with respect to arbitrary finite number of agents. A divide-and-conquer approach for cooperative tasking among multi-agent systems is proposed. The basic idea is to decompose an assigned global specification (given as a deterministic automaton) into subtasks for individual concurrent agents such that the fulfillment of these subtasks by each individual agent leads to the satisfaction of the global specification as a team. A cooperative scenario of three robots has been implemented to illustrate the proposed technique. This work provides insights on what kinds of tasks can be achieved distributively, which helps the designers to specify achievable global tasks for a group of agents and design necessary information sharing among each other for a particular task.

**Key Words:** Cooperative Control, Task Decomposition, Multi-agent Systems, Distributed Control, Discrete Event Systems

## I. INTRODUCTION

The study of multi-agent systems has emerged as a rapidly developing multi-disciplinary area with promising applications in assembling and transportation, parallel computing, distributed planning and scheduling, rapid emergency response and swarming robots [1]. The significance of multi-agent systems roots in the power of parallelism and cooperation between simple components that synergistically lead to sophisticated capabilities, robustness and functionalities [2]. The cooperative control of distributed multi-agent systems, however, is still in its infancy with significant practical and theoretical challenges that are difficult to be formulated and tackled by the traditional methods [3, 4]. Among these challenges, this paper

focuses on the cooperative tasking among a team of agents.

In the last decades, an extensive body of efforts has been devoted to the cooperative control of multi-agent systems. Examples of these results include consensus seeking [5, 6] and formation stabilization [7, 8], navigation functions for distributed formation [9], artificial potential functions [10], graph Laplacians for the associated neighborhood graphs [7, 11], graph-based formation stabilization and coordination [12, 13], distributed predictive control [14] and game theory-based coordinations [15]. These methods successfully model the interactions among the agents using the topology graph and apply Lyapunov-like energy functions and optimization methods for stabilization and formation of the continuous states of the agents. Most of the existing methods usually predefine the local interactions, e.g., the nearest neighbor law in consensus, attraction and impulsion forces due to local potential fields, and then investigate the emergent behavior collectively. However, a more relevant research question is how to design these local interactions to guarantee a given desired collective behavior or avoid undesired emergent behaviors.

---

M. Karimadini is with the Department of Electrical Engineering, Arak University of Technology, Arak, Iran, karimadini@arakut.ac.ir.

H. Lin is with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, USA. email: hlin1@nd.edu.

A. Karimoddini is with the Department of Electrical and Computer Engineering, North Carolina Agricultural and Technical State University, Greensboro, NC 27411 USA. email: akarimod@ncat.edu.

To address this problem, we aim to develop a provably correct-by-design method to synthesize local control laws and interaction rules among agents so as to achieve a desired global task collectively. While we focus on logical behaviors, it is assumed that the global task is given as a deterministic automaton and is broadcast to all agents. Individual agents however have limited access to the global information and project the global task automaton to their local event sets and obtain local specifications. The main research question here is that under what conditions the satisfactions of these local specifications distributively can imply the accomplishment of the global task. Toward this objective, in our previous work [16], the task decomposability condition was provided for the case of two agents and a hierarchical algorithm was proposed for more than two agents as a sufficient condition. The approach was then applied to the formation control of two unmanned helicopters [17]. This paper leverages our previous works by developing a scalable top-down correct-by-design method for distributed coordination and control of multi-agent systems such that the group of agents, as a team, can achieve the specified requirements, collectively. Necessary and sufficient conditions for the decomposability of a task automaton with respect to an arbitrary finite number of agents is provided. It is further shown that the fulfillment of local specifications can guarantee to achieve the global specification, provided the satisfaction of the decomposability conditions.

The proposed conditions serve beyond the direct comparison of the global task and the composition of local specifications [18], but gaining insights on the decomposability in terms of the task structure and event distribution among the agents. The proposed decomposability conditions state that an automaton is decomposable if and only if any decision on the order or selection between two transitions can be made by at least one of the agents, the interleaving of any pair of strings after synchronizing on a shared event does not introduce a new string that is not in the original automaton (the parallel composition of local task automata does not allow an illegal global behavior), and each local task automaton bisimulates a deterministic automaton (to ensure that the collection of local tasks does not disallow a legal global behavior). These insights may help operators to specify achievable global tasks for a group of agents and design necessary information sharing among each other for a particular task.

The rest of the paper is organized as follows. Preliminary notations, definitions and problem formulation are represented in Section II. Section III

introduces the necessary and sufficient conditions for the decomposability of an automaton with respect to parallel composition and an arbitrary finite number of local event sets. This section furthermore proves that, provided the task decomposability, the fulfillment of local tasks result in the satisfaction of global specification. This chapter also revisits the implementation of the cooperative control scenario of three robots [16] to illustrate the cooperative tasking for more than two agents. Finally, the paper concludes with remarks and discussions in Section IV. The proofs of lemmas are provided in the Appendix.

## II. PROBLEM FORMULATION

We first recall the definition of deterministic automaton [19].

**Definition 1** (*Automaton*) A deterministic automaton is a tuple  $A := (Q, q_0, E, \delta)$  consisting of a set of states  $Q$ ; an initial state  $q_0 \in Q$ ; a set of events  $E$  that causes transitions between the states, and a transition relation  $\delta \subseteq Q \times E \times Q$  (with a partial map  $\delta : Q \times E \rightarrow Q$ ), such that  $(q, e, q') \in \delta$  if and only if state  $q$  is transitioned to state  $q'$  by event  $e$ , denoted by  $q \xrightarrow{e} q'$  (or  $\delta(q, e) = q'$ ). In general the automaton also has an argument  $Q_m \subseteq Q$  of marked (accepting or final) states to assign a meaning of accomplishment to some states. For an automaton whose each state represents an accomplishment of a stage of the specification, all states can be considered as marked states and  $Q_m$  is omitted from the tuple.

With an abuse of notation, the definitions of the transition relation can be extended from the domain of  $Q \times E$  into the domain of  $Q \times E^*$  to define transitions over strings  $s \in E^*$ , where  $E^*$  stands for the Kleene – Closure of  $E$  (the collection of all finite sequences of events over elements of  $E$ ).

**Definition 2** (*Transition on strings*) For a deterministic automaton the existence of a transition over a string  $s \in E^*$  from a state  $q \in Q$  is denoted by  $\delta(q, s)!$  and inductively defined as  $\delta(q, \varepsilon) = q$ , and  $\delta(q, se) = \delta(\delta(q, s), e)$  for  $s \in E^*$  and  $e \in E$ . The existence of a set  $L \subseteq E^*$  of strings from a state  $q \in Q$  is then denoted as  $\delta(q, L)!$  and read as  $\forall s \in L : \delta(q, s)!$ .

The transition relation is a partial relation, and in general some of the states might not be accessible from the initial state.

**Definition 3** The operator  $Ac(\cdot)$  [20] is then defined by excluding the states and their attached transitions that are not reachable from the initial state as  $Ac(A) = (Q_{ac}, q_0, E, \delta_{ac})$  with  $Q_{ac} = \{q \in Q \mid \exists s \in E^*, q \in \delta(q_0, s)\}$  and  $\delta_{ac} = \{(q, e, q') \in \delta \mid e \in E, q, q' \in Q_{ac}\}$ . Since  $Ac(\cdot)$  has no effect on the behavior of the automaton, from now on we take  $A = Ac(A)$ .

The qualitative behavior of a deterministic system is described by its language defined as

**Definition 4** (Language, language equivalent automata) For a given automaton  $A$ , the language generated by  $A$  is defined as  $L(A) := \{s \in E^* \mid \delta(q_0, s)!\}$ . Two automata  $A_1$  and  $A_2$  are said to be language equivalent if  $L(A_1) = L(A_2)$ .

In cooperative tasking, each agent has a local observation from the global task: the perceived global task, filtered by its local event set, i.e., through a mapping over each agent's event set, as the interpretation of each agent from the global task. Particularly, natural projections  $P_{E_i}(A_S)$  are obtained from  $A_S$  by replacing its events that belong to  $E \setminus E_i$  by  $\varepsilon$ -moves, and then, merging the  $\varepsilon$ -related states. The  $\varepsilon$ -related states form equivalent classes defined as follows.

**Definition 5** (Equivalent class of states, [21]) Consider an automaton  $A_S = (Q, q_0, E, \delta)$  and an event set  $E' \subseteq E$ . Then, the relation  $\sim_{E'}$  is the minimal equivalence relation on the set  $Q$  of states such that  $q' \in \delta(q, e) \wedge e \notin E' \Rightarrow q \sim_{E'} q'$ , and  $[q]_{E'}$  denotes the equivalence class of  $q$  defined on  $\sim_{E'}$ . The set of equivalent classes of states over  $\sim_{E'}$ , is denoted by  $Q_{/\sim_{E'}}$  and defined as  $Q_{/\sim_{E'}} = \{[q]_{E'} \mid q \in Q\}$ .

$\sim_{E'}$  is an equivalence relation as it is reflective ( $q \sim_{E'} q$ ), symmetric ( $q \sim_{E'} q' \Leftrightarrow q' \sim_{E'} q$ ) and transitive ( $q \sim_{E'} q' \wedge q' \sim_{E'} q'' \Rightarrow q \sim_{E'} q''$ ).

It should be noted that the relation  $\sim_{E'}$  can be defined for any  $E' \subseteq E$ , for example,  $\sim_{E_i}$  and  $\sim_{E_i \cup E_j}$ , respectively denote the equivalence relations with respect to  $E_i$  and  $E_i \cup E_j$ . Moreover, when it is clear from the context,  $\sim_i$  is used to denote  $\sim_{E_i}$  for simplicity.

Next, natural projection over strings is denoted by  $p_{E'} : E^* \rightarrow E'^*$ , takes a string from the event set  $E$  and eliminates events in it that do not belong to the event set  $E' \subseteq E$ . The natural projection is formally defined on the strings as

**Definition 6** (Natural Projection on String, [20]) Consider a global event set  $E$  and an event set  $E' \subseteq E$ . Then, the natural projection  $p_{E'} : E^* \rightarrow E'^*$  is inductively defined as  $p_{E'}(\varepsilon) = \varepsilon$ , and  $\forall s \in E^*, e \in E$  :

$$p_{E'}(se) = \begin{cases} p_{E'}(s)e & \text{if } e \in E'; \\ p_{E'}(s) & \text{otherwise.} \end{cases}$$

The natural projection is then formally defined on an automaton as follows.

**Definition 7** (Natural Projection on Automaton) Consider a deterministic automaton  $A_S = (Q, q_0, E, \delta)$  and an event set  $E' \subseteq E$ . Then,  $P_{E'}(A_S) = (Q_i = Q_{/\sim_{E'}}, [q_0]_{E'}, E', \delta')$ , with  $[q']_{E'} \in \delta'([q]_{E'}, e)$  if there exist states  $q_1$  and  $q'_1$  such that  $q_1 \sim_{E'} q$ ,  $q'_1 \sim_{E'} q'$ , and  $\delta(q_1, e) = q'_1$ . Again,  $P_{E'}(A_S)$  can be defined into any event set  $E' \subseteq E$ . For example,  $P_{E_i}(A_S)$  and  $P_{E_i \cup E_j}(A_S)$ , respectively denote the natural projections of  $A_S$  into  $E_i$  and  $E_i \cup E_j$ . When it is clear from the context,  $P_{E_i}$  is replaced with  $P_i$ , for simplicity.

The collective task is then obtained using the parallel composition of local task automata, as the perception of the team from the global task.

**Definition 8** (Parallel Composition [19])

Let  $A_i = (Q_i, q_i^0, E_i, \delta_i)$ ,  $i = 1, 2$ , be automata. The parallel composition (synchronous composition) of  $A_1$  and  $A_2$  is the automaton  $A_1 \parallel A_2 = (Q = Q_1 \times Q_2, q_0 = (q_1^0, q_2^0), E = E_1 \cup E_2, \delta)$ , with  $\delta$  defined as  $\forall (q_1, q_2) \in Q, e \in E$  :

$$\delta((q_1, q_2), e) = \begin{cases} (\delta_1(q_1, e), \delta_2(q_2, e)), & \text{if } \delta_1(q_1, e)!, \delta_2(q_2, e)!, \\ & e \in E_1 \cap E_2; \\ (\delta_1(q_1, e), q_2), & \text{if } \delta_1(q_1, e)!, e \in E_1 \setminus E_2; \\ (q_1, \delta_2(q_2, e)), & \text{if } \delta_2(q_2, e)!, e \in E_2 \setminus E_1; \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

The parallel composition of  $A_i$ ,  $i = 1, 2, \dots, n$  is called parallel distributed system, and is defined based on the associativity property of parallel composition [20] as  $\parallel_{i=1}^n A_i := A_1 \parallel \dots \parallel A_n := A_n \parallel (A_{n-1} \parallel (\dots \parallel (A_2 \parallel A_1)))$ .

The obtained collective task is then compared with the original global task automaton using the bisimulation relation, in order to ensure that the team of agents understands the global specification, collectively.

**Definition 9** (Bisimulation [20]) Consider two automata  $A_i = (Q_i, q_i^0, E, \delta_i)$ ,  $i = 1, 2$ . The automaton  $A_1$  is said to be similar to  $A_2$  (or  $A_2$  simulates  $A_1$ ),

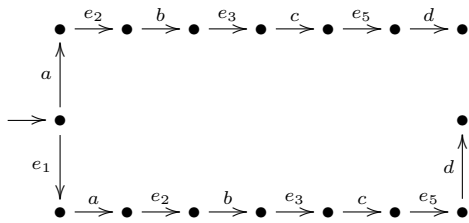
denoted by  $A_1 \prec A_2$ , if there exists a simulation relation from  $A_1$  to  $A_2$  over  $Q_1, Q_2$  and with respect to  $E$ , i.e., (1)  $(q_1^0, q_2^0) \in R$ , and (2)  $\forall (q_1, q_2) \in R, q_1' \in \delta_1(q_1, e)$ , then  $\exists q_2' \in Q_2$  such that  $q_2' \in \delta_2(q_2, e)$ ,  $(q_1', q_2') \in R$  [20].

Automata  $A_1$  and  $A_2$  are said to be bisimilar (bisimulate each other), denoted by  $A_1 \cong A_2$  if  $A_1 \prec A_2$  with a simulation relation  $R_1$ ,  $A_2 \prec A_1$  with a simulation relation  $R_2$  and  $R_1^{-1} = R_2$  [22], where  $R_1^{-1} = \{(y, x) \in Q_2 \times Q_1 | (x, y) \in R_1\}$ .

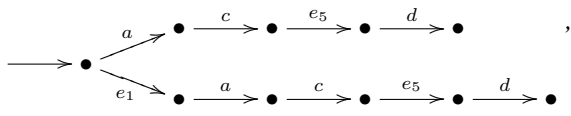
Based on these definitions we may now formally define the decomposability of an automaton with respect to parallel composition and natural projections as follows.

**Definition 10** (Automaton decomposability) A task automaton  $A_S$  with the event set  $E$  and local event sets  $E_i, i = 1, \dots, n$ ,  $E = \bigcup_{i=1}^n E_i$ , is said to be decomposable with respect to parallel composition and natural projections  $P_i, i = 1, \dots, n$ , when  $\prod_{i=1}^n P_i(A_S) \cong A_S$ .

**Example 1** The automaton  $A_S$ :



with  $E = E_1 \cup E_2 \cup E_3$ ,  $E_1 = \{a, c, d, e_1, e_5\}$ ,  $E_2 = \{a, b, d, e_2\}$ ,  $E_3 = \{b, c, e_3\}$ ,  $P_1(A_S)$ :



$P_2(A_S) \cong \rightarrow \bullet \xrightarrow{a} \bullet \xrightarrow{e_2} \bullet \xrightarrow{b} \bullet \xrightarrow{d} \bullet$  and  $P_3(A_S) \cong \rightarrow \bullet \xrightarrow{b} \bullet \xrightarrow{e_3} \bullet \xrightarrow{c} \bullet$ , is decomposable as  $A_S \cong P_1(A_S) || P_2(A_S) || P_3(A_S)$ .

**Remark 1** Since bisimilarity is an equivalence relation it is also transitive, and hence  $P_i(A_S)$ 's can be denoted as being bisimilar, rather than equal to the drawn automata, since  $P_i'(A_S) \cong P_i(A_S), i = 1, \dots, n$ , and  $\prod_{i=1}^n P_i'(A_S) \cong A_S$  is equivalent to  $\prod_{i=1}^n P_i(A_S) \cong A_S$ .

In [16], we proposed a necessary and sufficient condition for the task decomposability with respect to two agents. For more than two agents a hierarchical

algorithm was proposed to iteratively use the decomposability for two agents. The algorithm is a sufficient condition only, as it can decompose the task automaton if at each stage the task is decomposable with respect to one local event set and the rest of agents. For instance, in Example 1  $A_S$  is decomposable as  $A_S \cong P_1(A_S) || P_2(A_S) || P_3(A_S)$ , and choosing any of local event sets  $E_1, E_2$  and  $E_3$  the algorithm passes the first stage of hierarchical decomposition, as  $A_S \cong P_1(A_S) || (P_2(A_S) || P_3(A_S)) \cong P_3(A_S) || (P_1(A_S) || P_2(A_S)) \cong P_2(A_S) || (P_1(A_S) || P_3(A_S))$ , but it gets stuck at the second step, as  $P_{E_2 \cup E_3}(A_S) \not\cong P_2(A_S) || P_3(A_S)$ ,  $P_{E_1 \cup E_2}(A_S) \not\cong P_1(A_S) || P_2(A_S)$  and  $P_{E_1 \cup E_3}(A_S) \not\cong P_1(A_S) || P_3(A_S)$ . Moreover, it is possible to show by counterexamples that not all automata are decomposable with respect to parallel composition and natural projections (see following example). Then, a natural follow-up question is what makes an automaton decomposable.

**Problem 1** Given a deterministic task automaton  $A_S$  with event set  $E = \bigcup_{i=1}^n E_i$  and local event sets  $E_i, i = 1, \dots, n$ , what are the necessary and sufficient conditions that  $A_S$  is decomposable with respect to parallel composition and natural projections  $P_i, i = 1, \dots, n$ , such that  $\prod_{i=1}^n P_i(A_S) \cong A_S$ ?

### III. TASK DECOMPOSITION FOR $n$ AGENTS

The main result on task automaton decomposition is given as follows.

**Theorem 1** A deterministic automaton  $A_S = (Q, q_0, E = \bigcup_{i=1}^n E_i, \delta)$  is decomposable with respect to parallel composition and natural projections  $P_i, i = 1, \dots, n$  such that  $A_S \cong \prod_{i=1}^n P_i(A_S)$  if and only if

- DC1:  $\forall e_1, e_2 \in E, q \in Q: [\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\exists E_i \in \{E_1, \dots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1 e_2)! \wedge \delta(q, e_2 e_1)!]$ ;
- DC2:  $\forall e_1, e_2 \in E, q \in Q, s \in E^*: [\delta(q, e_1 e_2 s)! \vee \delta(q, e_2 e_1 s)!] \Rightarrow [\exists E_i \in \{E_1, \dots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1 e_2 s)! \wedge \delta(q, e_2 e_1 s)!]$ ;
- DC3:  $\forall q, q_1, q_2 \in Q$ , strings  $s, s'$  over  $E$ ,  $\delta(q, s) = q_1, \delta(q, s') = q_2, \exists i, j \in \{1, \dots, n\}, i \neq j, p_{E_i \cap E_j}(s), p_{E_i \cap E_j}(s')$  start with

$a \in E_i \cap E_j: \prod_{i=1}^n P_i(A) \prec_{A_S}(q)$  (where

$A := \begin{array}{c} \longrightarrow \bullet \xrightarrow{s} \bullet \\ \searrow_{s'} \bullet \end{array}$  and  $A_S(q)$  denotes an

automaton that is obtained from  $A_S$ , starting from  $q$ , and

- **DC4:**  $\forall i \in \{1, \dots, n\}, x, x_1, x_2 \in Q_i, x_1 \neq x_2, e \in E_i, t \in E_i^*, x_1 \in \delta_i(x, e), x_2 \in \delta_i(x, e): \delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$

*Proof:* In order for  $A_S \cong \prod_{i=1}^n P_i(A_S)$ , from the definition of bisimulation, it is required to have  $A_S \prec \prod_{i=1}^n P_i(A_S)$ ;  $\prod_{i=1}^n P_i(A_S) \prec A_S$ , and the simulation relations are inverse of each other. These requirements are provided by the following three lemmas.

Firstly,  $\prod_{i=1}^n P_i(A_S)$  always simulates  $A_S$ . Formally:

**Lemma 1** Consider a deterministic automaton  $A_S = (Q, q_0, E = \bigcup_{i=1}^n E_i, \delta)$  and natural projections  $P_i, i = 1, \dots, n$ . Then, it always holds that  $A_S \prec \prod_{i=1}^n P_i(A_S)$ .

The similarity of  $\prod_{i=1}^n P_i(A_S)$  to  $A_S$ , however, is not always true (see Example 2), and needs some conditions as stated in the following lemma.

**Lemma 2** Consider a deterministic automaton  $A_S = (Q, q_0, E = \bigcup_{i=1}^n E_i, \delta)$  and natural projections  $P_i, i = 1, \dots, n$ . Then,  $\prod_{i=1}^n P_i(A_S) \prec A_S$  if and only if

- **DC1:**  $\forall e_1, e_2 \in E, q \in Q: [\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\exists E_i \in \{E_1, \dots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1 e_2)! \wedge \delta(q, e_2 e_1)!]$ ;
- **DC2:**  $\forall e_1, e_2 \in E, q \in Q, s \in E^*: [\delta(q, e_1 e_2 s)! \vee \delta(q, e_2 e_1 s)!] \Rightarrow [\exists E_i \in \{E_1, \dots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1 e_2 s)! \wedge \delta(q, e_2 e_1 s)!]$ ;
- **DC3:**  $\forall q, q_1, q_2 \in Q$ , strings  $s, s'$  over  $E$ ,  $\delta(q, s) = q_1, \delta(q, s') = q_2, \exists i, j \in \{1, \dots, n\}, i \neq j, p_{E_i \cap E_j}(s), p_{E_i \cap E_j}(s')$  start with

$a \in E_i \cap E_j: \prod_{i=1}^n P_i(A) \prec_{A_S}(q)$  (where

$A := \begin{array}{c} \longrightarrow \bullet \xrightarrow{s} \bullet \\ \searrow_{s'} \bullet \end{array}$  and  $A_S(q)$  is an

automaton that is obtained from  $A_S$ , starting from  $q$ ).

Next, we need to show that for two simulation relations  $R_1$  (for  $A_S \prec \prod_{i=1}^n P_i(A_S)$ ) and  $R_2$  (for

$\prod_{i=1}^n P_i(A_S) \prec A_S$ ) defined by the above two lemmas,  $R_1^{-1} = R_2$ .

**Lemma 3** Consider an automaton  $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$  with natural projections  $P_i, i = 1, \dots, n$ .

If  $A_S$  is deterministic,  $A_S \prec \prod_{i=1}^n P_i(A_S)$  with the

simulation relation  $R_1$  and  $\prod_{i=1}^n P_i(A_S) \prec A_S$  with the

simulation relation  $R_2$ , then  $R_1^{-1} = R_2$  (i.e.,  $\forall q \in Q, z \in Z: (z, q) \in R_2 \Leftrightarrow (q, z) \in R_1$ ) if and only if **DC4:**  $\forall i \in \{1, \dots, n\}, x, x_1, x_2 \in Q_i, x_1 \neq x_2, e \in E_i, t \in E_i^*, x_1 \in \delta_i(x, e), x_2 \in \delta_i(x, e): \delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$

Now, Theorem 1 is proven as follows. Firstly, conditions **DC1** and **DC2** in Theorem 1 are equivalent to the respective conditions in Lemma 2 due to the logical equivalences  $(p \wedge q) \Rightarrow r \equiv q \Rightarrow (\neg p \vee r)$  and  $p \Leftrightarrow q \equiv (p \vee q) \Rightarrow (p \wedge q)$ , for any expressions  $p$  and  $q$ . Then, according to Definition 9,  $A_S \cong \prod_{i=1}^n P_i(A_S)$  if

and only if  $A_S \prec \prod_{i=1}^n P_i(A_S)$  (that is always true due to

Lemma 1),  $\prod_{i=1}^n P_i(A_S) \prec A_S$  (that it is true if and only

if **DC1**, **DC2** and **DC3** are true, according to Lemma 2) and  $R_1^{-1} = R_2$  (that for a deterministic automaton

$A_S$ , when  $A_S \prec \prod_{i=1}^n P_i(A_S)$  with simulation relation

$R_1$  and  $\prod_{i=1}^n P_i(A_S) \prec A_S$  with simulation relation  $R_2$ ,

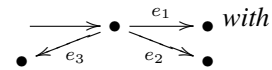
due to Lemma 3,  $R_1^{-1} = R_2$  holds true if and only if

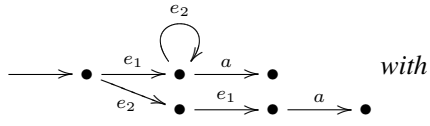
**DC4** is satisfied). Therefore,  $A_S \cong \prod_{i=1}^n P_i(A_S)$  if and only if **DC1**, **DC2**, **DC3** and **DC4** are satisfied. ■

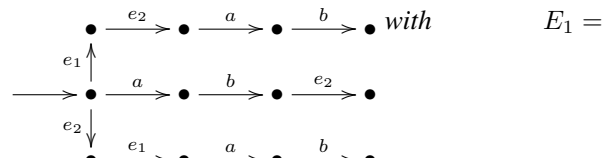
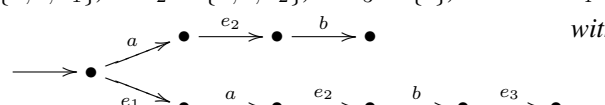
**Remark 2** Intuitively, the decomposability condition **DC1** means that for any decision on the selection between two transitions there should exist at least one agent that is capable of the decision making, or the decision should not be important (both permutations in any order be legal). **DC2** says that for any decision on the order of two successive events before any string, either there should exist at least one agent capable of such decision making, or the decision should not be important, i.e., any order would be legal for occurrence of that string. The condition **DC3** means that the interleaving of strings from local task automata

that share the first appearing shared event ( $p_{E_i \cap E_j}(s)$  and  $p_{E_i \cap E_j}(s')$  start with the same event  $a \in E_i \cap E_j$ ), should not allow a string that is not allowed in the original task automaton. In other words, DC3 is to ensure that an illegal behavior (a string that does not appear in  $A_S$ ) is not allowed by the team (does not appear in  $\prod_{i=1}^n P_i(A_S)$ ). The last condition, DC4, deals with the nondeterminism of local automata. Here,  $A_S$  is deterministic, whereas  $P_i(A_S)$  could be nondeterministic. DC4 ensures the determinism of bisimulation quotient of local task automata, in order to guarantee that the simulation relations from  $A_S$  to  $\prod_{i=1}^n P_i(A_S)$  and vice versa are inverse of each other. By providing this property, DC4 guarantees that a legal behavior (appearing in  $A_S$ ) is not disabled by the team (appears in  $\prod_{i=1}^n P_i(A_S)$ ).

Example 1 showed a decomposable automaton. Following example illustrate the automata that are indecomposable due to violation of one of the decomposability conditions DC1-DC4, respectively, although satisfy other three conditions.

**Example 2** The automata  $A_1$ :  with local event sets  $E_1 = \{e_1, e_3\}$ ,  $E_2 = \{e_2\}$ ,  $E_3 = \{e_3\}$ ;

$A_2$ :  with  $E_1 = \{a, e_1\}$ ,  $E_2 = \{a, e_2\}$ ;

$A_3$ :  with  $E_1 = \{a, b, e_1\}$ ,  $E_2 = \{a, b, e_2\}$ ,  $E_3 = \{b\}$ , and  $A_4$ :  with

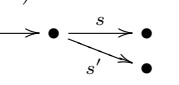
$E = E_1 \cup E_2 \cup E_3$ ,  $E_1 = \{a, b, e_1, e_2, e_3\}$ ,  $E_2 = E_3 = \{a, b, e_2, e_3\}$  are not decomposable as they respectively do not satisfy DC1, DC2, DC3 and DC4, while fulfill other three conditions.

**Remark 3** (Decidability of the conditions) Since this work deals with finite state automata, the expression  $s \in E^*$  in the decomposability conditions can be checked over finite states as follows.

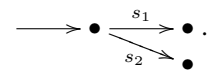
The first condition DC1 involves no expression “ $s \in E^*$ ”, and hence, can be checked over the finite number of states and transitions. According to the definition, the second condition DC2:  $\forall e_1, e_2 \in E, q \in Q, s \in E^*, \forall E_i \in \{E_1, \dots, E_n\}, \{e_1, e_2\} \not\subseteq E_i$ :  $\delta(q, e_1 e_2 s)! \Leftrightarrow \delta(q, e_2 e_1 s)!$ ; (or DC2:  $\forall e_1, e_2 \in E, q \in Q, s \in E^*$ :  $[\delta(q, e_1 e_2 s)! \vee \delta(q, e_2 e_1 s)!] \Rightarrow [\exists E_i \in \{E_1, \dots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1 e_2 s)! \wedge \delta(q, e_2 e_1 s)!]$ ) can be checked by showing the existence of a relation  $\hat{R}_2$  on the states reachable from  $\delta(q, e_1 e_2)$  and  $\delta(q, e_2 e_1)$  as  $(\delta(q, e_1 e_2), \delta(q, e_2 e_1)) \in \hat{R}_2, \forall (q_1, q_2) \in \hat{R}_2, e \in E$ :

1.  $\delta(q_1, e) = q'_1 \Rightarrow \exists q'_2 \in Q, \delta(q_2, e) = q'_2, (q'_1, q'_2) \in \hat{R}_2$ , and
2.  $\delta(q_2, e) = q'_2 \Rightarrow \exists q'_1 \in Q, \delta(q_1, e) = q'_1, (q'_1, q'_2) \in \hat{R}_2$ .

For instance,  $A_2$  in Example 2 violates DC2 as  $(\delta(q_0, e_1 e_2), \delta(q_0, e_2 e_1)) \in \hat{R}_2, \exists e_2 \in E, \delta(\delta(q_0, e_1 e_2), e_2)!, \text{ but } \neg \delta(\delta(q_0, e_2 e_1), e_2)!$ .

Checking DC3 also can be done over finite states by corresponding the pairs of strings  $s, s'$  such that  $\exists q, q_1, q_2 \in Q, \delta(q, s) = q_1, \delta(q, s') = q_2, \exists i, j \in \{1, \dots, n\}, i \neq j, p_{E_i \cap E_j}(s), p_{E_i \cap E_j}(s')$  start with  $a \in E_i \cap E_j$ , and then forming  $A :=$  

and  $A_S(q)$  (an automaton that is obtained from  $A_S$ , starting from  $q$ ). and checking  $\prod_{i=1}^n P_i(A) \prec A_S(q)$ . For example, consider  $A_3$  in Example 2 and let  $s_1, s_2$  and  $s_3$  denote its strings from top to bottom. This automaton is not decomposable since  $\prod_{i=1}^n P_i(A) \not\prec A_S(q_0)$  for  $A :=$

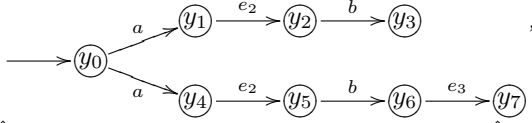
. Here,  $s_1$  and  $s_2$  synchronize on  $a \in$

$E_1 \cap E_2$  and generate a new string  $e_1 a b e_2$  in  $\prod_{i=1}^n P_i(A)$  that does not appear in  $A_S$ . The fourth condition (DC4:  $\forall i \in \{1, \dots, n\}, x, x_1, x_2 \in Q_i, x_1 \neq x_2, e \in E_i, t \in E_i^*, x_1 \in \delta_i(x, e), x_2 \in \delta_i(x, e): \delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$ ) also can be checked over finite states, by checking the existence of a relation  $\hat{R}_4$  on the states reachable from  $x_1$  and  $x_2$  as  $(x_1, x_2) \in \hat{R}_4, \forall (x_3, x_4) \in \hat{R}_4, e \in E$ :

1.  $x'_3 \in \delta_i(x_3, e) \Rightarrow \exists x'_4 \in Q_i, x'_4 \in \delta_i(x_4, e), (x'_3, x'_4) \in \hat{R}_4$ , and
2.  $x'_4 \in \delta_i(x_4, e) \Rightarrow \exists x'_3 \in Q_i, x'_3 \in \delta_i(x_3, e), (x'_3, x'_4) \in \hat{R}_4$ .

Definition of this relation is a direct implication of DC4 that requires identical strings after any

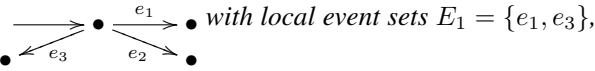
nondeterministic transition in any local automaton. For example, the task automaton  $A_4$  in Example 2 does not satisfy DC4, as for  $P_2(A_S) \cong P_3(A_S) \cong$



$\hat{R}_4 = \{(y_1, y_4), (y_2, y_5), (y_3, y_6)\}$ ,  $(y_3, y_6) \in \hat{R}_4$ ,  $\exists e_3 \in E$ ,  $\delta_2(y_6, e_3)!$ , but  $\neg \delta_2(y_3, e_3)!$ .

More importantly, the proposed method provides some guideline on the structure of the global specification automaton and the distribution the events among the agents in order for decomposability.

**Remark 4** (Insights on enforcing the decomposability conditions) The result in Theorem 1 provides us some hints for ruling out indecomposable task automata and for enforcing the violated decomposability conditions. For example, if  $\exists e_1, e_2 \in E, q \in Q$ :  $[\delta(q, e_1)! \wedge \delta(q, e_2)!]$  but neither  $\exists E_i \in \{E_1, \dots, E_n\}, \{e_1, e_2\} \subseteq E_i$  nor  $\delta(q, e_1 e_2)! \wedge \delta(q, e_2 e_1)!$ , then  $A_S$  is not decomposable due to the violation of DC1. To remove this violation there should exist an agent with local event set  $E_i \in \{E_1, \dots, E_n\}$  such that  $\{e_1, e_2\} \subseteq E_i$ . For instance, in the automata  $A_S$ :

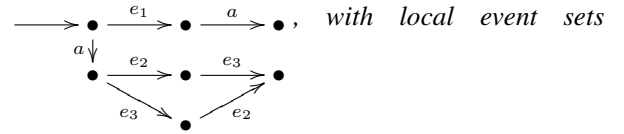


with local event sets  $E_1 = \{e_1, e_3\}$ ,  $E_2 = \{e_2\}$ ,  $E_3 = \{e_3\}$ , if  $E_2 = \{e_1, e_2\}$  and  $E_3 = \{e_2, e_3\}$ , then DC1 was satisfied. This solution also works for an indecomposability of  $A_S$  due to a violation of DC2 where  $\exists e_1, e_2 \in E, q \in Q, s \in E^*$ :  $\delta(q, e_1 e_2 s)! \vee \delta(q, e_2 e_1 s)!$  but neither  $\exists E_i \in \{E_1, \dots, E_n\}, \{e_1, e_2\} \subseteq E_i$  nor  $\delta(q, e_1 e_2 s)! \wedge \delta(q, e_2 e_1 s)!$ . For example, in  $A_S$ :

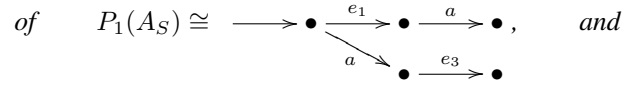


with local event sets  $E_1 = \{e_1, e_3\}$ ,  $E_2 = \{e_2\}$ ,  $E_3 = \{e_3\}$ , again if  $E_2 = \{e_1, e_2\}$  and  $E_3 = \{e_2, e_3\}$ , then DC1 was satisfied. Violation of other two conditions, DC3 and DC4, is caused due to synchronization of two different branches  $s$  and  $s'$  from different local task automata, say  $P_i(A_S)$  and  $P_j(A_S)$ , on a common event  $a \in E_i \cap E_j$ . This synchronization may impose an ambiguity in understanding of  $A_S$ , when  $P_i(A_S)$  and  $P_j(A_S)$  synchronize on  $a$ . If one string in  $P_i(A_S)$  after synchronization on  $a$ , continues to another string in  $P_j(A_S)$  and this interleaving generates a new string in  $\prod_{i=1}^n P_i(A_S)$  that does not appear in  $A_S$ , then DC3 is dissatisfied, whereas if this interleaving causes that a string in  $A_S$  cannot be completed

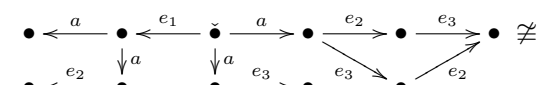
in  $\prod_{i=1}^n P_i(A_S)$ , then DC4 is violated. DC4 can be also violated due to a nondeterminism on a private event in a local automaton, which again causes an ambiguity in the collective task  $\prod_{i=1}^n P_i(A_S)$ . One way to remove this ambiguity is therefore by introducing the first events in  $s$  and  $s'$  to both  $E_i$  and  $E_j$ . In this case the synchronization on event  $a$  will only occur on the projections of identical strings from  $A_S$  and also it avoids the nondeterminism in local automata. For example, the task automaton  $A_S$ :



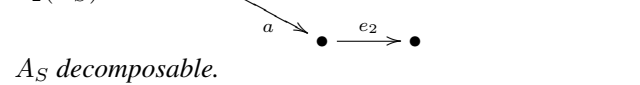
$E_1 = \{a, e_1, e_3\}$  and  $E_2 = \{a, e_2\}$  satisfies DC1 and DC2, but violates DC3 and DC4, and hence is not decomposable as the parallel composition of  $P_1(A_S) \cong$



and  $P_2(A_S) \cong$



is  $P_1(A_S) \parallel P_2(A_S) \cong$



$A_S$  decomposable.

Once the task is decomposed into local tasks and the local controllers are designed for each local plant, the next question is guaranteeing the global specification, provided each local closed loop system satisfies its corresponding local specification.

The cooperative tasking result can be now presented as follows.

**Theorem 2** Consider a plant, represented by a parallel distributed system  $\prod_{i=1}^n A_{P_i}$ , with given local event sets  $E_i$ ,  $i = 1, \dots, n$ , and let the global specification is given by a deterministic task automaton  $A_S$ , with  $E = \bigcup_{i=1}^n E_i$ . Then, designing local controllers  $A_{C_i}$ , so that  $A_{C_i} \parallel A_{P_i} \cong P_i(A_S)$ ,  $i = 1, \dots, n$ , derives the global closed loop system to satisfy the global specification  $A_S$ , in the sense of bisimilarity, i.e.,  $\prod_{i=1}^n (A_{C_i} \parallel A_{P_i}) \cong A_S$ , provided DC1, DC2, DC3 and DC4 for  $A_S$ .

*Proof:* Following two lemmas are presented to be used for the proof.

**Lemma 4** (Associativity of parallel composition [20])  
 $P_1(A_S) \parallel P_2(A_S) \parallel \cdots \parallel P_{n-1}(A_S) \parallel P_n(A_S) \cong$   
 $P_n(A_S) \parallel (P_{n-1}(A_S) \parallel (\cdots \parallel (P_2(A_S) \parallel P_1(A_S))))$ .

**Lemma 5** [16] *If two automata  $A_2$  and  $A_4$  (bi)simulate, respectively,  $A_1$  and  $A_3$ , then  $A_2 \parallel A_4$  (bi)simulates  $A_1 \parallel A_3$ , i.e.,*

1.  $(A_1 \prec A_2) \wedge (A_3 \prec A_4) \Rightarrow$   
 $(A_1 \parallel A_3 \prec A_2 \parallel A_4)$ ;
2.  $(A_1 \cong A_2) \wedge (A_3 \cong A_4) \Rightarrow$   
 $(A_1 \parallel A_3 \cong A_2 \parallel A_4)$ .

Now, satisfying DC1-DC4 for  $A_S$ , according to Theorem 1, leads to decomposability of  $A_S$  into local task automata  $P_i(A_S)$ ,  $i = 1, \dots, n$ , such that  $A_S \cong \prod_{i=1}^n P_i(A_S)$ . Then, choosing local controllers  $A_{C_i}$ , so that  $A_{C_i} \parallel A_{P_i} \cong P_i(A_S)$ ,  $i = 1, 2, \dots, n$ , due to Lemma 5.2, results in  $\prod_{i=1}^n (A_{C_i} \parallel A_{P_i}) \cong \prod_{i=1}^n P_i(A_S) \cong A_S$ . ■

In the following example, we recall the task automaton of cooperative multi-robot scenario from [16] (with the correction of robot indices  $R_2$ ,  $R_1$  and  $R_3$  from right to the left), where the global task automaton has been decomposed into local task automata using a hierarchical approach as a sufficient condition by which the decomposability conditions for 2 agents are successively used for  $n$  agents. Here, we decompose  $A_S$  directly using Theorem 1.

**Example 3** (Revisiting Example in Section 5 for decomposability using Theorem 1) Consider a team of three robots  $R_1$ ,  $R_2$  and  $R_3$  in Figure 1, initially in Room 1. All doors are equipped with spring to be closed automatically, when there is no force to keep them open. After a help announcement from Room 2, the Robot  $R_2$  is required to go to Room 2, urgently from the one-way door  $D_2$  and accomplish its task there and come back immediately to Room 1 from the two-way, but heavy door  $D_1$  that needs the cooperation of two robots  $R_1$  and  $R_3$  to be opened. To save time, as soon as the robots hear the help request from Room 2,  $R_2$  and  $R_3$  go to Rooms 2 and 3, from  $D_2$  and the two-way door  $D_3$ , respectively, and then  $R_1$  and  $R_3$  position on  $D_1$ , synchronously open  $D_1$  and wait for the accomplishment of the task of  $R_2$  in Room 2 and returning to Room 1 ( $R_2$  is fast enough). Afterwards,  $R_1$  and  $R_3$  move backward to close  $D_1$

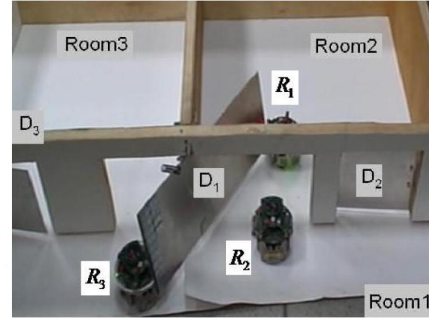
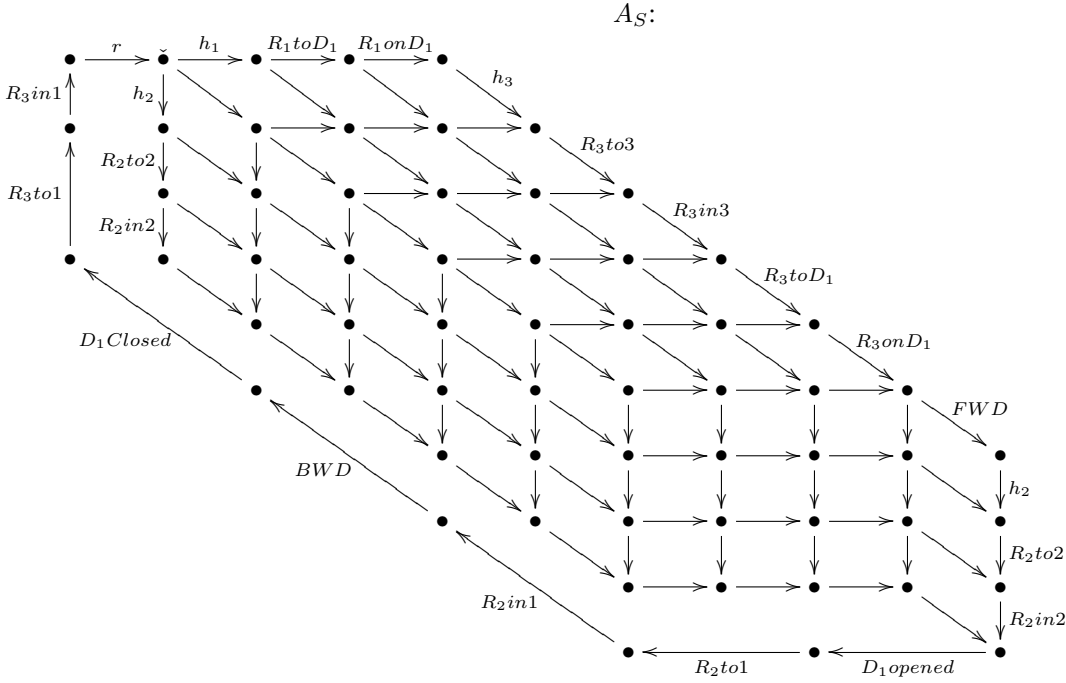
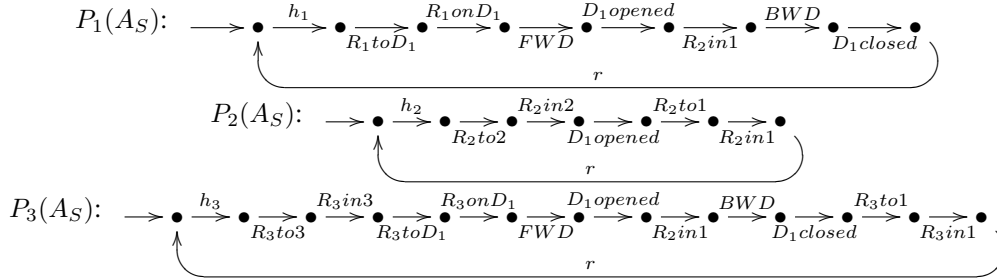


Fig. 1. The environment of MRS coordination example.

and then  $R_3$  returns back to Room 1 from  $D_3$ . All robots then stay at Room 1 for the next task [16]. These requirements can be translated into a task automaton for the robot team as it is illustrated in Figure 2, defined over local event sets  $E_1 = \{h_1, R_1toD_1, R_1onD_1, FWD, D_1opened, R_2in1, BWD, D_1closed, r\}$ ,  $E_2 = \{h_2, R_2to2, R_2in2, D_1opened, R_2to1, R_2in1, r\}$ , and  $E_3 = \{h_3, R_3to3, R_3in3, R_3toD_1, R_3onD_1, FWD, D_1opened, R_2in1, BWD, D_1closed, R_3to1, R_3in1, r\}$ , with  $h_i := R_i$  received help request,  $i = 1, 2, 3$ ;  $R_jtoD_1 :=$  command for  $R_j$  to position on  $D_1$ ,  $j = 1, 3$ ;  $R_jonD_1 := R_j$  has positioned on  $D_1$ ,  $j = 1, 3$ ;  $FWD :=$  command for moving forward (to open  $D_1$ );  $BWD :=$  command for moving backward (to close  $D_1$ );  $D_1opened := D_1$  has been opened;  $D_1closed := D_1$  has been closed;  $r :=$  command to go to initial state for the next implementation;  $R_itok :=$  command for  $R_i$  to go to Room  $k$ , and  $R_ink := R_i$  has gone to Room  $k$ ,  $i = 1, 2, 3$ ,  $k = 1, 2, 3$ .

To check the decomposability of  $A_S$  using Theorem 1, firstly DC1 and DC2 are satisfied since for any order/selection on the pairs events, each from one of the sets  $\{h_1, R_1toD_1, R_1onD_1\} \subseteq E_1 \setminus \{E_2 \cup E_3\}$ ,  $\{h_2, R_2to2, R_2in2\} \subseteq E_2 \setminus \{E_1 \cup E_3\}$  and  $\{h_3, R_3to3, R_3in3, R_3toD_1, R_3onD_1\} \subseteq E_3 \setminus \{E_1 \cup E_2\}$  and also the pairs of event  $FW$ , paired with events from  $\{h_2, R_2to2, R_2in2\}$ , the events appear in both orders in the automaton. The rest of orders/selections on transitions that are not legal in both orders can be decided by at least one agent, as  $\{R_1onD_1, FWD\} \subseteq E_1$ ,  $\{R_3onD_1, FWD\} \subseteq E_3$ ,  $\{FWD, D_1opened\} \subseteq E_1$ ,  $\{D_1opened, R_2to1\} \subseteq E_2$ ,  $\{R_2to1, R_2in1\} \subseteq E_2$ ,  $\{R_2in1, BWD\} \subseteq E_1$ ,  $\{BWD, D_1closed\} \subseteq E_1$ ,  $\{D_1closed, R_3to1\} \subseteq E_3$ ,  $\{R_3to1, R_3in1\} \subseteq E_3$ ,  $\{R_3in1, r\} \subseteq E_3$ ,  $\{r, h_1\} \subseteq E_1$ ,  $\{r, h_2\} \subseteq E_2$ ,  $\{r, h_3\} \subseteq E_3$ . Moreover, since starting from any



Fig. 2. Task automaton  $A_S$  for robot team.Fig. 3.  $P_1(A_S)$  for  $R_1$ ;  $P_2(A_S)$  for  $R_2$  and  $P_3(A_S)$  for  $R_3$ .

state, each shared event  $e \in \{FWD, D_1opened, R_2in1, BWD, D_1closed, r\}$  appears in only one branch, DC3 is satisfied. Furthermore, DC4 is also satisfied since  $P_i(A_S)$ ,  $i = 1, 2, 3$  are deterministic automata. Therefore, according to Theorem 1,  $A_S$  is decomposable into  $P_i(A_S)$ ,  $i = 1, 2, 3$ , as illustrated in Figure 3, bisimulates  $A_S$ .

Choosing local controllers  $AC_i := P_i(A_S)$  leads to  $AC_i \parallel AP_i \cong P_i(A_S)$ ,  $i = 1, 2, 3$  that according to Theorem 2 results in  $\parallel_{i=1}^n (AC_i \parallel AP_i) \cong \parallel_{i=1}^n P_i(A_S) \cong A_S$ , i.e., the team of controlled robots collectively satisfy the global specification  $A_S$ . The scenario has

been successfully implemented on a team of three ground robots.

#### IV. CONCLUSION

The paper proposed a formal method for automaton decomposability, applicable in top-down decentralized cooperative control of distributed discrete event systems. Given a set of agents whose logical behaviors are modeled in a parallel distributed system, and a global task automaton, the paper has the following contributions: firstly, we provide necessary and sufficient conditions for decomposability of an

automaton with respect to parallel composition and natural projections into an arbitrary finite number of local event sets, and secondly, it has been shown that if a global task automaton is decomposed for individual agents, designing a local supervisor for each agent, satisfying its local task, guarantees that the closed loop system of the team of agents satisfies the global specification.

The proposed decomposability conditions can be applied to the discrete event systems in which all states are marked. The example of such systems include the manufacturing machines with routine tasks, execution of PLC (programmable Logic Controller) systems that the subroutines are visited iteratively, and any other such systems that all states of the system should be visited and hence can be attributed to marked states. Therefore, future works include the extension of the results for the class of systems with only some of the states as marked states. For this purpose new decomposability conditions have to be developed such that the composition of local automata preserves the marked states of the global task automaton. Other interesting directions on this topic are the fault-tolerant task decomposition in spite of failure in some events, and decomposabilizability of an indecomposable task automaton by modifying the event distribution.

## Appendix A. Definitions

This part provides some definitions to be used during the proofs of the lemmas in the Appendix. Firstly, we successive event pair and adjacent event pair are defined as follows.

**Definition 11** (Successive event pair) Two events  $e_1$  and  $e_2$  are called successive events if  $\exists q \in Q : \delta(q, e_1)! \wedge \delta(\delta(q, e_1), e_2)! \text{ or } \delta(q, e_2)! \wedge \delta(\delta(q, e_2), e_1)!$ .

**Definition 12** (Adjacent event pair) Two events  $e_1$  and  $e_2$  are called adjacent events if  $\exists q \in Q : \delta(q, e_1)! \wedge \delta(q, e_2)!$ .

We will also use synchronized product of languages in the following section, defined as follows.

**Definition 13** (Synchronized product of languages [23]) Consider a global event set  $E$  and local event sets  $E_i$ ,  $i = 1, \dots, n$ , such that  $E = \bigcup_{i=1}^n E_i$ . For a finite set of languages  $\{L_i \subseteq E_i^*\}_{i=1}^n$ , the synchronized product (product language) of  $\{L_i\}$ , denoted by  $\prod_{i=1}^n L_i$ , is

defined as  $\prod_{i=1}^n L_i = \{s \in E^* \mid \forall i \in \{1, \dots, n\} : p_i(s) \in L_i\} = \bigcap_{i=1}^n p_i^{-1}(L_i)$ .

**Remark 5** Using the product language, it is then possible to characterize the language of parallel composition of two automata  $A_1$  and  $A_2$ , with respective event sets  $E_1$  and  $E_2$ , in terms of their languages, as  $L(A_1 \parallel A_2) = L(A_1) \parallel L(A_2) = p_1^{-1}(L(A_1)) \cap p_2^{-1}(L(A_2))$  with  $p_i : (E_1 \cup E_2)^* \rightarrow E_i^*$ ,  $i = 1, 2$  [23]. Accordingly, the interleaving of two strings is defined as the product language to their respective automata as follows. Let  $A_1 = (\{q_1, \dots, q_n\}, \{q_1\}, E_1 = \{e_1, \dots, e_n\}, \delta_1)$  and  $A_2 = (\{q'_1, \dots, q'_m\}, \{q'_1\}, E_2 = \{e'_1, \dots, e'_m\}, \delta_2)$  denote path automata (automata with only one branch)  $q_1 \xrightarrow{e_1}$

$q_2 \xrightarrow{e_2} \dots \xrightarrow{e_n} q_n$  and  $q'_1 \xrightarrow{e'_1} q'_2 \xrightarrow{e'_2} \dots \xrightarrow{e'_m} q'_m$ , respectively. Then,  $L(A_1 \parallel A_2) = \bar{s} \mid \bar{s}' = p_1^{-1}(\bar{s}) \cap p_2^{-1}(\bar{s}')$  with  $s = e_1, \dots, e_n$ ,  $s' = e'_1, \dots, e'_m$  and  $p_i : (E_1 \cup E_2)^* \rightarrow E_i^*$ ,  $i = 1, 2$ . Here,  $\bar{s}$  denotes the prefix-closure of an string, defined as the set of all prefixes of the string. Formally, if  $s$  is the event sequence  $s := e_1 e_2 \dots e_n$ , then  $\bar{s} := \{\varepsilon, e_1, e_1 e_2, \dots, e_1 e_2 \dots e_n\}$ .

**Example 4** Consider three strings  $s_1 = e_1 a$ ,  $s_2 = a e_2$  and  $s_3 = a e_1$ . Then the interleaving of  $s_1$  and  $s_2$  is  $\bar{s}_1 \mid \bar{s}_2 = \bar{e}_1 a \bar{e}_2$  while the interleaving of two strings  $s_2$  and  $s_3$  becomes  $\bar{s}_2 \mid \bar{s}_3 = \{a e_1 e_2, a e_2 e_1\}$ .

## Appendix B. Proof for Lemma 1

Recalling Lemma 1 in [16], stating that for a deterministic automaton  $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$ ,  $A_S \prec P_1(A_S) \parallel P_2(A_S)$ , it leads to  $P_{\bigcup_{i=m}^n E_i}(A_S) \prec P_m(A_S) \parallel P_{\bigcup_{i=m+1}^n E_i}(A_S)$ ,  $m = 1, \dots, n-1$ , for  $A_S = (Q, q_0, E = \bigcup_{i=1}^n E_i, \delta)$ . Therefore,  $A_S \cong P_{\bigcup_{i=1}^n E_i}(A_S) \prec P_1(A_S) \parallel P_{\bigcup_{i=2}^n E_i}(A_S) \prec P_1(A_S) \parallel P_2(A_S) \parallel P_{\bigcup_{i=3}^n E_i}(A_S) \prec \dots \prec \prod_{i=1}^n P_i(A_S)$ .

## Appendix C. Proof for Lemma 2

**Sufficiency:** Consider the deterministic automaton  $A_S = (Q, q_0, E, \delta)$ . The set of transitions in  $\prod_{i=1}^n P_i(A_S) = (Z, z_0, E, \delta_{\parallel})$  is defined as  $T = \{z_0 :=$

$(x_0^1, \dots, x_0^n) \xrightarrow{i=1}^n \overline{p_i(s_i)} z := (x_1, \dots, x_n) \in Z := \prod_{i=1}^n Q_i$ , where,  $(x_0^1, \dots, x_0^n) \xrightarrow{i=1}^n \overline{p_i(s_i)} (x_1, \dots, x_n)$  in  $\prod_{i=1}^n P_i(A_S)$  is the interleaving of strings  $x_0^i \xrightarrow{p_i(s_i)} x_i$  in  $P_i(A_S)$ ,  $i = 1, \dots, n$  (projections of  $q_0 \xrightarrow{s_i} \delta(q_0, s_i)$  in  $A_S$ ). Let  $\tilde{L}(A_S) \subseteq L(A_S)$  denote the largest subset of  $L(A_S)$  such that  $\forall s \in \tilde{L}(A_S), \exists s' \in \tilde{L}(A_S), \exists E_i, E_j \in \{E_1, \dots, E_n\}, i \neq j, p_{E_i \cap E_j}(s)$  and  $p_{E_i \cap E_j}(s')$  start with the same event. Then,  $T$  can be divided into three sets of transitions corresponding to a division of  $\{\Gamma_1, \Gamma_2, \Gamma_3\}$  on the set of interleaving strings  $\Gamma = \left\{ \prod_{i=1}^n \overline{p_i(s_i)} \mid s_i \in E^*, q_0 \xrightarrow{s_i} \delta(q_0, s_i) \right\}$ , where,  $\Gamma_1 = \left\{ \prod_{i=1}^n \overline{p_i(s_i)} \in \Gamma \mid s_1, \dots, s_n \notin \tilde{L}(A_S), s_1 = \dots = s_n \right\}$ ,  $\Gamma_2 = \left\{ \prod_{i=1}^n \overline{p_i(s_i)} \in \Gamma \mid s_1, \dots, s_n \notin \tilde{L}(A_S), \exists s_i, s_j \in \{s_1, \dots, s_n\}, s_i \neq s_j \right\}$ ,  $\Gamma_3 = \left\{ \prod_{i=1}^n \overline{p_i(s_i)} \in \Gamma \mid s_i \in \tilde{L}(A_S) \right\}$ . Moreover, since  $A_S$  is deterministic,  $\prod_{i=1}^n P_i(A_S) \prec A_S$  is reduced to  $\delta(q_0, \prod_{i=1}^n \overline{p_i(s)})!$  in  $A_S$  for transitions in  $\Gamma$ .  $\prod_{i=1}^n P_i(A_S) \prec A_S$ .

Thus, defining a relation  $R$  as  $(z_0, q_0) \in R, R := \{(z, q) \in Z \times Q \mid \exists t \in E^*, z \in \delta_{\parallel}(z_0, t)\}$ , the aim is to show that  $R$  is a simulation relation from  $\prod_{i=1}^n P_i(A_S)$  to  $A_S$ .

For the interleavings in  $\Gamma_1$ ,  $\forall z, z_1 \in Z, e \in E, z_1 \in \delta_{\parallel}(z, e): \exists q, q_1 \in Q, \delta(q, e) = q_1$  such that  $\forall z[j] \in \{z[1], \dots, z[n]\}$  (the  $j$ -th component of  $z$ ),  $\exists l \in \text{loc}(e), z[j] = [q]_l$ . Similarly,  $\forall e' \in E, z_2 \in Z, z_2 \in \delta_{\parallel}(z_1, e'): \exists q_2 \in Q, \delta(q_1, e') = q_2$ . Now, if  $\nexists E_i \in \{E_1, \dots, E_n\}, \{e, e'\} \in E_i$ , then the definition of parallel composition will furthermore induce that  $\exists z_3 \in Z, z_3 \in \delta_{\parallel}(z, e'), z_2 \in \delta_{\parallel}(z_3, e)$ . This, together with  $DC1$  and  $DC2$  implies that  $\exists q_3, q_4 \in Q, \delta(q, e') = q_3, \delta(q_3, e) = q_4$  and that  $\forall t \in E^*, \delta_{\parallel}(z_2, t)!: \delta(q_2, t)!$  and  $\delta(q_4, t)!$ . Therefore, any path automaton in  $\prod_{i=1}^n P_i(A_S)$  is simulated by  $A_S$ , and hence,  $\delta(q_0, \prod_{i=1}^n \overline{p_i(s)})!$  in  $A_S, \forall s \in L(A_S)$ .

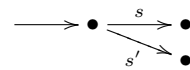
For the interleavings in  $\Gamma_2$ , from the definition of  $\Gamma_2$ , it follows that for any set of  $s_i, \delta(q_0, s_i)!, i \in \{1, \dots, n\}$ , two cases are possible for  $\Gamma_2$ :

**Case 1:**  $\forall s, s' \in \{s_1, \dots, s_n\}, \forall E_i, E_j \in \{E_1, \dots, E_n\}: p_{E_i \cap E_j}(s) = \varepsilon$  and  $p_{E_i \cap E_j}(s') = \varepsilon$ . In this case, projections of such strings  $s_i$  can be written as  $p_i(s_i) = e_1^i, \dots, e_{m_i}^i, i = 1, \dots, n$ . The interleaving of these projected strings leads to a grid of states and transitions in  $\prod_{i=1}^n \prod_{j_i=0}^{m_i} x_{j_i}^i$

as  $(x_{j_1}^1, \dots, x_{j_n}^n) \xrightarrow{e_j^i} (y_{j_1}^1, \dots, y_{j_n}^n)$ , with  $y_{j_i}^{i_k} = \begin{cases} x_{j_{i+1}}^{i_k}, & \text{if } i = i_k, j = j_i + 1 \\ x_{j_i}^{i_k}, & \text{otherwise} \end{cases} j_i = 0, 1, \dots, m_i, i = 1, \dots, n, i_k = 1, \dots, n, k = 1, \dots, n$ . This grid of transitions is simulated by counterpart transitions in  $A_S$ , as  $\forall s, s' \in \{s_1, \dots, s_n\}$ , for any two successive/adjacent events  $e_j^i$  and  $e_{j'}^{i'}$ , both orders exist in  $A_S$ , due to  $DC1$  and  $DC2$ , and hence,  $\delta(q_{j_i, i_k}, e_j^k) = q'_{j_i, i_k}, j_i = 0, 1, \dots, m_i, i = 1, \dots, n, i_k = 1, \dots, n, k = 1, \dots, n$ . Therefore, for any choice of  $s_i$  corresponding to  $\Gamma_2, \delta(q_0, \prod_{i=1}^n \overline{p_i(s_i)})!$  in  $A_S$ .

**Case 2:**  $\exists s, s' \in \{s_1, \dots, s_n\}, \exists E_i, E_j \in \{E_1, \dots, E_n\}: p_{E_i \cap E_j}(s) \neq \varepsilon$  or  $p_{E_i \cap E_j}(s') \neq \varepsilon$ , but they do not start with the same event. Any such  $s$  and  $s'$  can be written as  $s = t_1 a t_2$  and  $s' = t_1' b t_2'$ , where  $t_1 = e_1 \dots e_m, t_1' = e_1' \dots e_{m'}' \notin (E_i \cap E_j)^*, \forall i, j \in \{1, \dots, n\}, i \neq j, \exists i, j \in \{1, \dots, n\}, i \neq j, a, b \in E_i \cap E_j, t_2, t_2' \in E^*$ . Therefore, due to synchronization constraint, the interleaving of strings will not evolve from  $a$  and  $b$  onwards, and hence,  $\overline{p_i(s)} \overline{p_j(s')} = \overline{p_i(t_1)} \overline{p_j(t_1')}$  and  $\overline{p_i(s')} \overline{p_j(s)} = \overline{p_i(t_1')} \overline{p_j(t_1)}$ , and Case 2 is reduced to Case 1, leading to  $\delta(q_0, \prod_{i=1}^n \overline{p_i(s_i)})!$  in  $A_S$ .

Furthermore, due to  $DC3$ , for any two distinct strings  $s, s' \in \tilde{L}(A_S)$  (i.e., two strings starting from state  $q$  in  $A_S$  that  $\exists E_i, E_j \in \{E_1, \dots, E_n\}, i \neq j, p_{E_i \cap E_j}(s), p_{E_i \cap E_j}(s')$  start with the same event  $a \in E_i \cap E_j$ ) we have  $\prod_{i=1}^n P_i(A) \prec A_S(q)$  (where  $A :=$



that is obtained from  $A_S$ , starting from  $q$ ). This is particularly true for  $q = q_0$ . Therefore,  $DC3$  implies that for the pair of strings  $s, s'$  (over the transitions in  $\Gamma_3$ ), and corresponding automaton  $A, L(\prod_{i=1}^n P_i(A)) \subseteq L(A_S)$ , that from the definition of synchronized product means that  $\prod_{i=1}^n p_i^{-1}(\{\bar{s}, \bar{s}'\}) \subseteq L(A_S)$ . For any pair of  $s', s'' \in \tilde{L}(A_S)$  also  $DC3$  similarly results in  $\prod_{i=1}^n p_i^{-1}(\{\bar{s}', \bar{s}''\}) \subseteq L(A_S)$ , that collectively results

in  $\bigcap_{i=1}^n p_i^{-1}(\{\bar{s}, \bar{s}', \bar{s}''\}) \subseteq L(A_S)$ , due to the following lemma:

**Lemma 6** [20] For any two languages  $L_1, L_2$  defined over an event set  $E$  and a natural projection  $p : E^* \rightarrow E_i^*$ , for  $E_i \subseteq E$ :  $p_i(L_1 \cup L_2) = p_i(L_1) \cup p_i(L_2)$  and  $p_i^{-1}(L_1 \cup L_2) = p_i^{-1}(L_1) \cup p_i^{-1}(L_2)$ .

This, inductively means that for  $\{s_1 \cdots, s_m\} \subseteq \tilde{L}(A_S)$ :  $\bigcap_{i=1}^n p_i^{-1}(\{s_i\}_{i=1}^m) \subseteq L(A_S)$ , i.e.,  $\delta(q_0, \big|_{i=1}^n \overline{p_i(s_i)})!$  in  $A_S$ , for transitions in  $\Gamma_3$ .

Therefore, *DC3* implies that all transitions in  $\Gamma$  are simulated by transitions in  $A_S$  that because of the determinism of  $A_S$  results in  $\big|_{i=1}^n P_i(A_S) \prec A_S$ .

**Necessity:** The necessity is proven by contradiction. Assume that  $\big|_{i=1}^n P_i(A_S) \prec A_S$ , but *DC1*, *DC2* or *DC3* is not satisfied.

If *DC1* is violated, then  $\exists e_1, e_2 \in E$ ,  $q \in Q$ ,  $\nexists E_i \in \{E_1, \dots, E_n\}$ ,  $\{e_1, e_2\} \subseteq E_i$ ,  $[\delta(q, e_1)! \wedge \delta(q, e_2)!] \wedge \neg[\delta(q, e_1 e_2)! \wedge \delta(q, e_2 e_1)!]$ .

However,  $\delta(q, e_1)! \wedge \delta(q, e_2)!$ , from the definition of natural projection, implies that  $\delta_i([q]_i, e_1)! \wedge \delta_j([q]_j, e_2)!$ , in  $P_i(A_S)$  and  $P_j(A_S)$ , respectively,  $\forall i \in \text{loc}(e_1), j \in \text{loc}(e_2)$ . This in turn, from definition of parallel composition leads to  $\delta_{\parallel}([q]_1, \dots, [q]_n, e_1)!$  and  $\delta_{\parallel}([q]_1, \dots, [q]_n, e_2)!$ .

This means that  $\delta_{\parallel}([q]_1, \dots, [q]_n, e_1 e_2)! \wedge \delta_{\parallel}([q]_1, \dots, [q]_n, e_2 e_1)!$  in  $\big|_{i=1}^n P_i(A_S)$ , but  $\neg[\delta(q, e_1 e_2)! \wedge \delta(q, e_2 e_1)!]$  in  $A_S$ , i.e.,  $\big|_{i=1}^n P_i(A_S) \not\prec A_S$  which contradicts with the hypothesis.

If *DC2* is not satisfied, then  $\exists e_1, e_2 \in E$ ,  $q \in Q$ ,  $\nexists E_i \in \{E_1, \dots, E_n\}$ ,  $\{e_1, e_2\} \subseteq E_i$ ,  $s \in E^*$ ,  $\neg[\delta(q, e_1 e_2 s)! \Leftrightarrow \delta(q, e_2 e_1 s)!]$ , i.e.,  $[\delta(q, e_1 e_2 s)! \vee \delta(q, e_2 e_1 s)!] \wedge \neg[\delta(q, e_1 e_2 s)! \wedge \delta(q, e_2 e_1 s)!]$ . The expression  $[\delta(q, e_1 e_2 s)! \vee \delta(q, e_2 e_1 s)!]$  from definition of natural projection and Lemma 1, respectively implies that  $\delta_{\parallel}([q]_1, \dots, [q]_n, e_1 e_2 s)!$  and  $\delta_{\parallel}([q]_1, \dots, [q]_n, e_2 e_1 s)!$  and  $\delta_{\parallel}([q]_1, \dots, [q]_n, e_1 e_2 s)! \wedge \delta_{\parallel}([q]_1, \dots, [q]_n, e_2 e_1 s)!$  in  $\big|_{i=1}^n P_i(A_S)$ . This in turn leads to

$\delta_{\parallel}([q]_1, \dots, [q]_n, e_1 e_2 s)! \wedge \delta_{\parallel}([q]_1, \dots, [q]_n, e_2 e_1 s)!$  in  $\big|_{i=1}^n P_i(A_S)$ , but  $\neg[\delta(q, e_1 e_2 s)! \wedge \delta(q, e_2 e_1 s)!]$  in  $A_S$ , that contradicts with  $\big|_{i=1}^n P_i(A_S) \prec A_S$ .

$\delta_{\parallel}([q]_1, \dots, [q]_n, e_1 e_2 s)! \wedge \delta_{\parallel}([q]_1, \dots, [q]_n, e_2 e_1 s)!$  in  $\big|_{i=1}^n P_i(A_S)$ , but  $\neg[\delta(q, e_1 e_2 s)! \wedge \delta(q, e_2 e_1 s)!]$  in  $A_S$ , that contradicts with  $\big|_{i=1}^n P_i(A_S) \prec A_S$ .

The violation of *DC3* also leads to contradiction as  $\delta(q_0, s_i)!$ ,  $i = 1, \dots, n$ , results in  $\delta_{\parallel}([q_0]_1, \dots, [q_0]_n, \big|_{i=1}^n \overline{p_i(s_i)})!$  in  $\big|_{i=1}^n P_i(A_S)$ , whereas  $\neg\delta(q_0, \big|_{i=1}^n \overline{p_i(s_i)})!$  in  $A_S$ .

## Appendix D. Proof for Lemma 3

**Sufficiency:** Following two lemmas are used in the proof of Lemma 3.

**Lemma 7** (Lemma 9 in [16]) Consider two automata  $A_1$  and  $A_2$ , and let  $A_1$  be deterministic,  $A_1 \prec A_2$  with the simulation relation  $R_1$  and  $A_2 \prec A_1$  with the simulation relation  $R_2$ . Then,  $R_1^{-1} = R_2$  if and only if there exists a deterministic automaton  $A'_1$  such that  $A'_1 \cong A_2$ .

Next, let  $A_1$  and  $A_2$  be substituted by  $A_S$  and  $\big|_{i=1}^n P_i(A_S)$ , respectively, in Lemma 7. Then, the existence of  $A'_1 = A'_S$  in Lemma 7 is characterized by the following lemma.

**Lemma 8** Consider a deterministic automaton  $A_S$  and its natural projections  $P_i(A_S)$ ,  $i = 1, \dots, n$ . Then, there exists a deterministic automaton  $A'_S$  such that  $A'_S \cong \big|_{i=1}^n P_i(A_S)$  if and only if there exist deterministic automata  $P'_i(A_S)$  such that  $P'_i(A_S) \cong P_i(A_S)$ ,  $i = 1, \dots, n$ .

**Proof:** Let  $A_S = (Q, q_0, E = \bigcup_{i=1}^n E_i, \delta)$ ,  $P_i(A_S) = (Q_i, q_0^i, E_i, \delta_i)$ ,  $P'_i(A_S) = (Q'_i, q_0'^i, E_i, \delta'_i)$ ,  $i = 1, \dots, n$ ,  $\big|_{i=1}^n P_i(A_S) = (Z, z_0, E, \delta_{\parallel})$ ,  $\big|_{i=1}^n P'_i(A_S) = (Z', z_0', E, \delta'_{\parallel})$ . Then, the proof of Lemma 8 is presented as follows.

**Sufficiency:** The existence of deterministic automata  $P'_i(A_S)$  such that  $P'_i(A_S) \cong P_i(A_S)$ ,  $i = 1, \dots, n$  implies that  $\delta'_i$ ,  $i = 1, \dots, n$  are functions, and consequently from definition of parallel composition (Definition 8),  $\delta'_{\parallel}$  is a function, and hence  $\big|_{i=1}^n P'_i(A_S)$  is deterministic. Moreover, from Lemma 5,  $P'_i(A_S) \cong P_i(A_S)$ ,  $i = 1, \dots, n$  lead to  $\big|_{i=1}^n P'_i(A_S) \cong \big|_{i=1}^n P_i(A_S)$ , meaning that there exists a deterministic automaton  $A'_S := \big|_{i=1}^n P'_i(A_S)$  such that  $A'_S \cong \big|_{i=1}^n P_i(A_S)$ .

**Necessity:** The necessity is proven by contraposition, namely, by showing that if there does not exist deterministic automata  $P'_i(A_S)$  such that  $P'_i(A_S) \cong P_i(A_S)$ , for  $i = 1, 2, \dots, n$ , then there does not exist a deterministic automaton  $A'_S$  such that  $A'_S \cong \prod_{i=1}^n P_i(A_S)$ .

Without loss of generality, assume that there does not exist a deterministic automaton  $P'_1(A_S)$  such that  $P'_1(A_S) \cong P_1(A_S)$ . This means that  $\exists q, q_1, q_2 \in Q$ ,  $e \in E_1$ ,  $t_1, t_2 \in (E \setminus E_1)^*$ ,  $t \in E^*$ ,  $\delta(q, t_1 e) = q_1$ ,  $\delta(q, t_2 e) = q_2$ ,  $\neg[\delta(q_1, t)! \Leftrightarrow \delta(q_2, t)!]$ , meaning that  $\delta(q_1, t)! \wedge \neg\delta(q_2, t)!$  or  $\neg\delta(q_1, t)! \wedge \delta(q_2, t)!$ . Again without loss of generality we consider the first case and show that it leads to a contradiction. The contradiction of the second case is followed, similarly. From the first case,  $\delta(q_1, t)! \wedge \neg\delta(q_2, t)!$ , definition of natural projection, definitions of parallel composition and Lemma 1 it follows that  $([q_1]_1, ([q_1]_2, \dots, [q_1]_n)) \in \delta_{||}([q_1], ([q_2], \dots, [q_n]), t_1 e)$ ,  $([q_2]_1, ([q_1]_2, \dots, [q_1]_n)) \in \delta_{||}([q_1], ([q_2], \dots, [q_n]), t_1 e)$ ,  $\delta([q_1]_1, ([q_1]_2, \dots, [q_1]_n), t)!$ , whereas  $\neg\delta([q_2]_1, ([q_1]_2, \dots, [q_1]_n), t)!$  in  $\prod_{j=1}^n P_j(A_S)$ , implying that there does not exist a deterministic automaton  $A'_S$  such that  $A'_S \cong \prod_{j=1}^n P_j(A_S)$ , and the necessity is followed. ■

Now, Lemma 3 is proven as follows.

**Sufficiency:** *DC4* implies that there exist deterministic automata  $P'_i(A_S)$  such that  $P'_i(A_S) \cong P_i(A_S)$ ,  $i = 1, \dots, n$ . Then, from Lemmas 5 and 8, it follows, respectively, that  $\prod_{i=1}^n P'_i(A_S) \cong \prod_{i=1}^n P_i(A_S)$ , and that there exists a deterministic automaton  $A'_S := \prod_{i=1}^n P'_i(A_S)$  such that  $A'_S \cong \prod_{i=1}^n P_i(A_S)$  that due to Lemma 7, it results in  $R_1^{-1} = R_2$ .

**Necessity:** Let  $A_S$  be deterministic,  $A_S \prec \prod_{i=1}^n P_i(A_S)$  with the simulation relation  $R_1$  and  $\prod_{i=1}^n P_i(A_S) \prec A_S$  with the simulation relation  $R_2$ , and assume by contradiction that  $R_1^{-1} = R_2$ , but *DC4* is not satisfied. Violation of *DC4* implies that for  $\exists i \in \{1, \dots, n\}$ , there does not exist a deterministic automaton  $P'_i(A_S)$  such that  $P'_i(A_S) \cong P_i(A_S)$ . Therefore, due to Lemma 8, there does not exist a deterministic automaton  $A'_S$  such that  $A'_S \cong \prod_{i=1}^n P_i(A_S)$ , and hence, according to Lemma 7, it leads to  $R_1^{-1} \neq R_2$  which is a contradiction.

## REFERENCES

1. F. Bullo, J. Cortés, S. Martinez, Distributed control of robotic networks: a mathematical approach to motion coordination algorithms, Princeton Univ Pr, 2009.
2. R. Murray, Recent research in cooperative control of multivehicle systems, Journal of Dynamic Systems, Measurement, and Control 129 (2007) 571.
3. P. Tabuada, G. Pappas, Linear time logic control of discrete-time linear systems, Automatic Control, IEEE Transactions on 51 (12) (2006) 1862–1877. doi:10.1109/TAC.2006.886494.
4. M. Kloetzer, C. Belta, Temporal logic planning and control of robotic swarms by hierarchical abstractions, Robotics, IEEE Transactions on 23 (2) (2007) 320–330. doi:10.1109/TRO.2006.889492.
5. Y. Xin, Z. Cheng, Consensus control in linear multi-agent systems with current and sampled partial relative states, Asian Journal of Control 16 (4) (2014) 1105–1111. doi:10.1002/asjc.775.
6. J. Wang, D. Cheng, X. Hu, Consensus of multi-agent linear dynamic systems, Asian Journal of Control 10 (2) (2008) 144–155. doi:10.1002/asjc.15.
7. A. Jadbabaie, J. Lin, A. Morse, Coordination of groups of mobile autonomous agents using nearest neighbor rules, Automatic Control, IEEE Transactions on 48 (6) (2003) 988–1001. doi:10.1109/TAC.2003.812781.
8. A. Karimodini, H. Lin, Hierarchical hybrid symbolic robot motion planning and control, Asian Journal of Control. doi:DOI:10.1002/asjc.1027.
9. S. G. Loizou, K. J. Kyriakopoulos, Multirobot Navigation Functions II: Towards Decentralization, Book Series Studies in H. A. P Blom and J. Lygeros (Eds.), Stochastic Hybrid Systems: Theory and Safety Critical Applications, Springer, 2006.
10. E. Rimon, D. Koditschek, Exact robot navigation using artificial potential functions, Robotics and Automation, IEEE Transactions on 8 (5) (1992) 501–518.
11. H. Tanner, A. Jadbabaie, G. Pappas, Flocking in fixed and switching networks, Automatic Control, IEEE Transactions on 52 (5) (2007) 863–868.
12. J. Fax, R. Murray, Information flow and cooperative control of vehicle formations, Automatic

- Control, IEEE Transactions on 49 (9) (2004) 1465–1476.
13. H. Tanner, G. Pappas, V. Kumar, Leader-to-formation stability, Robotics and Automation, IEEE Transactions on 20 (3) (2004) 443–455.
  14. A. Richards, J. How, Decentralized model predictive control of cooperating uavs, in: Decision and Control, 2004. CDC. 43rd IEEE Conference on, Vol. 4, IEEE, 2004, pp. 4286–4291.
  15. E. Semsar-Kazerooni, K. Khorasani, Multi-agent team cooperation: A game theory approach, Automatica 45 (10) (2009) 2205–2213.
  16. M. Karimadini, H. Lin, Guaranteed global performance through local coordinations, Automatica 47 (5) (2011) 890 – 898.
  17. A. Karimodini, M. Karimadini, H. Lin, Decentralized hybrid formation control of unmanned aerial vehicles, in: American Control Conference (ACC), 2014, 2014, pp. 3887–3892.
  18. M. Mukund, From global specifications to distributed implementations, in B. Caillaud, P. Darondeau, L. Lavagno (Eds.), *Synthesis and Control of Discrete Event Systems*, Kluwer, Springer Berlin / Heidelberg, Berlin, 2002.
  19. R. Kumar, V. K. Garg, Modeling and Control of Logical Discrete Event Systems, Kluwer Academic Publishers, Norwell, MA, USA, 1999.
  20. C. G. Cassandras, S. Lafortune, Introduction to discrete event systems, Springer, USA, 2008.
  21. R. Morin, Decompositions of asynchronous systems, in: CONCUR '98: Proceedings of the 9th International Conference on Concurrency Theory, Springer-Verlag, London, UK, 1998, pp. 549–564.
  22. F. Liu, H. Lin, Z. Dziong, Bisimilarity control of partially observed nondeterministic discrete event systems and a test algorithm, Automatica 47 (4) (2011) 782788.
  23. Y. Willner, M. Heymann, Supervisory control of concurrent discrete-event systems, International Journal of Control 54 (1991) 1143–1169.