Cooperative Tasking for Multi-agent Systems

Mohammad Karimadini

NATIONAL UNIVERSITY OF SINGAPORE

2011

Cooperative Tasking for Multi-agent Systems

Mohammad Karimadini

(*M.Sc., University Putra Malaysia*)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2011

In the name of Allah, Most Gracious, Most Merciful

“Guide us to the straight path.”

“Read in the name of your lord who created, created the human from a (blood) clot.
Read! your lord is the most generous,
who taught by the pen, taught the human what he did not know. ”

“Holy Quran”

I present this thesis to my parents, parents in law, wife, sons, brothers, sisters and all relatives, friends and teachers who thought me how to feel, think, learn and apply.

Acknowledgements

First of all thanks to my god, Allah, who has continuously provided my heart strengths, passion and guidance to pursue my PhD successfully, and also peace upon prophet Mohammad (SAW), his family and followers.

First and foremost, I would like to gratefully thank my supervisor Professor Hai Lin for his great supervision, patience, encouragement and kindness. Without his guidance, this thesis would not have been possible. I also thank Professor Tong Heng Lee and Professor Xiang Cheng for their valuable comments during my comprehensive and oral qualifying exams and Professor Ben M. Chen and Professor Xiang Cheng for agreeing to be my thesis committee; all lecturers in ECE Department and former teachers who have built my academic background, and all ECE and NUS staff and laboratory officers for their official supports. Special thanks also to Mr. Kaveh Khalilpour and Madam Khalida Ramzan Khan for helping me to settle in Singapore. I had also a wonderful time with all of my friends in campus life as well as academic life, especially Mr. Ali Karimoddini, Mr. Mohsen Zamani, Mr. Alireza Partovi, Ms. Sun Yajuan, Prof. Liu Fuchun, Dr. Yang Yang, Ms. Li Xiaoyang, Mr. Liu Xiaomeng, Ms. Xue Zhengui, Mr. Yao Jin and Mr. Mohammad Reza Chamanbaz. I also highly appreciate the FYP students Mr. Wong Jinsheng and Mr. Mohamed Isa Bin Mohamed Nasser for their help on the implementation of multi-robot scenario for cooperative tasking.

I also thank my parents (Mr. Mohammad Mehdi and Ms. Mones) and parents in law (Mr. Mohammad Reza and Ms. Farokh) for their support and valuing my dreams; my beloved wife (Ms. Atefeh) and twin sons (Mr. Arash & Mr. Kaveh) and all relatives and friends for their company along this journey and sharing their love to me as the source of my inspiration.

Contents

Summary

It is an amazing fact that remarkably complex behaviors could emerge from a large collection of very rudimentary dynamical agents through very simple local interactions. However, it still remains elusive on how to design these local interactions among agents so as to achieve certain desired collective behaviors. This thesis aims to tackle this challenge and proposes a divide-and-conquer approach to guarantee specified global behaviors through local coordination and control design for multi-agent systems. The basic idea is to decompose a requested global specification into subtasks for each individual agent such that the fulfillment of these subtasks by each individual agent leads to the satisfaction of the global specification as a team. For this purpose, three issues are studied here: (1) task decomposition for top-down design, such that the fulfillment of local tasks guarantees the satisfaction of the global task, by the team; (2) fault-tolerant top-down design, such that the global task remains decomposable and achievable, in spite of some failures, and (3) the design of interactions among agents to make an indecomposable task decomposable and achievable in the top-down framework.

To address the first question, namely the cooperative tasking of multi-agent systems, it is firstly shown by a counterexample that not all specifications can be decomposed in this sense. Then, necessary and sufficient conditions are identified for the decomposability of a task for two cooperative agents; based on decision making on the orders and selections of transitions, the interleaving of synchronized strings and the determinism of local tasks. The decomposability conditions are then generalized to the case of arbitrary finite number of agents, and furthermore, it is shown that the fulfillment of local specifications can guarantee the satisfaction of the global specification. Finally, a cooperative control scenario for a team of three robots is developed to illustrate the task decomposition procedure.

The thesis then deals with the robustness issues of the proposed top-down design approach with respect to event failures in the multi-agent system. The main concern under event failure is whether a previously decomposable task can still be achieved collectively by the agents and if not, we would like to investigate that under what conditions the global task could be robustly accomplished. This is actually the fault-tolerance issue of the top-down design, and the results provide designers with hints on which events are fragile with respect to failures, and whether redundancies are needed. The main objective of this part of the work is to identify conditions on failed events under which a decomposable global task can still be achieved, successfully. For such a purpose, a notion called passivity is introduced to characterize the type of event failures. The passivity is found to reflect the redundancy of communication links over shared events, based on which necessary and sufficient conditions for the reliability of the task decomposability and conditions on cooperative tasking under event failures are derived.

As the next important question, the thesis aims to study whether one can modify

the communication pattern between agents so as to make an indecomposable task decomposable. In particular, we would like to ask what is exactly needed to share by communication among agents such that an originally indecomposable task becomes decomposable. To answer this question, the decomposability conditions are revisited and all possible causes for indecomposibility are identified. As the main contribution of this part, the thesis then proposes a procedure, as a sufficient condition, to make an indecomposable deterministic task automaton decomposable in order to facilitate the cooperative tasking.

This result may pave the way towards a new perspective for the decentralized cooperative control of multi-agent systems.

List of Figures

Chapter 1

Introduction

1.1 Multi-agent Systems

1.1.1 Motivation and Background

Multi-agent system has emerged as a fascinating research area with strong attention from a wide range of applications such as distributed plants (power grids, sensor networks, transportation systems, distributed control, distributed planning and scheduling, distributed supply chains), distributed computational systems (decentralized optimization, parallel processing, concurrent computing, cloud computing) and multi-robot systems. Multi-agent system is therefore a developing multi-disciplinary area across various fields such as control engineering [1], computer sciences [2] and robotics [3–10]. The significance of multi-agent systems roots in the power of parallelism and cooperation between simple components that synergistically lead to sophisticated capabilities and more robustness and functionalities than individual multi-skilled agents [3, 11]. Cooperative control of multi-agent systems is therefore of great importance to the society and demands new methods and frameworks to analyze and design the interaction rules and control laws among the agents.

The cooperative control of a large number of dynamical agents, however, is in the infancy stage and possesses significant theoretical and practical challenges such as coordination, reconfiguration, synchronization and sequencing of the tasks that may fall beyond the conventional methodologies [12, 13]. On the other hand, the cooperation of agents provides new opportunities to achieve sophisticated team specifications. In multi-agent systems, each agent is usually equipped with certain, but very limited, degrees of sensing, processing, communication, and maneuvering capabilities. However, a large collection of such rudimentary systems, as a team, can display high level of functionalities and exhibit complex collective behaviors [11, 14–17]. These findings generate increasing motivations towards more research efforts on the area of cooperative control of multi-agent systems. In the next part we briefly review some of the existing methods in multi-agent systems and investigate them from the structural point of view.

1.1.2 Existing methods

Existing methods in multi-agent systems have been mainly developed based on heuristical, empirical and simulation studies, and mostly focused on bottom-up approaches to understand how and what global behaviors can be generated from simple local interactions [18, 19]. Usually, these local interaction rules are through biomimicry and draw inspirations from the swarming behaviors of biological systems, such as colonies of ants, hives of bees, flocks of birds, and schools of fishes [18, 20, 21]. As a result, it has been widely accepted that complicated collective behaviors can be emerged from a large collection of simple agents via intuitive local interaction rules [14–16].

Recently, further serious efforts have been devoted to developing rigorous theoretical frameworks for multi-agent systems. Remarkable efforts have been devoted to the consensus seeking and formation stabilization [22–25], while approaches like navigation functions [26–28] and artificial potential functions [29, 30] for distributed formation, graph Laplacians for the associated neighborhood graphs [23, 31], optimization-based path planning [32–34], parallel processing [35, 36], bottom-up task sharing and planning [37, 38], game theory-based coordinations [39], distributed learning [40], and geometrical swarming [17, 41] have been developed in the literature.

1.1.3 Top-down Versus Bottom-up Approaches

Although we know that sophisticated collective behaviors could emerge from a large collection of very elementary agents through simple local interactions, we still lack knowledge on how to change these rules to achieve or avoid certain global behaviors. As a result, it still remains elusive on how to design local interactions between the agents to make sure that they, as a group, can achieve the specified requirements. The desired global specification could be very sophisticated, and the design may go beyond the traditional output regulation, path planning, or formation control [12, 13, 42, 43]. Moreover, the bottom-up scenario in swarming robotics may fail to guarantee the correctness by design, due to the lack of understanding on how to manipulate the local rules to achieve the global behavior. To remove the undesirable global behaviors we may therefore need to redesign the local coordination rules through an iterative trial and error process, which may quickly become inefficient or even intractable for practical applications. This problem demands a new and formal method to design the local control laws and interaction rules for agents, directly, such that the desired global specification can be guaranteed by design. In particular, this thesis aims at developing a 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 (Figure 1.1).

For this purpose, the thesis proposes a divide-and-conquer design for cooperative multi-agent systems so as to guarantee the desired global behaviors. The core idea is to decompose a global specification into sub-specifications for individual agents, and then design local controllers for each agent to satisfy these local specifications, respectively. The decomposition should be done in such a way that the global behavior is achieved provided that all these sub-specifications are held true by individual agents. Hence, the global specification is guaranteed by design. In order to perform this idea, several questions are required to be answered as will be discussed in the next sections of this chapter. These questions include how to describe the global specification and subtasks in a succinct and formal way; how to decompose the global specification (i.e., how to obtain local tasks, how to compose them and how to compare the composed one with the original global task); whether it is always possible to decompose the task, and if not what are the necessary and sufficient conditions for decomposability and how to enforce them. In top-down design, when the logical behavior of the team is concerned, the global specification can be modeled by discrete event systems. Next part will discuss the specifications in the top-down supervisory control of logical behaviors, with the emphasize on the automaton model that is very close to the human

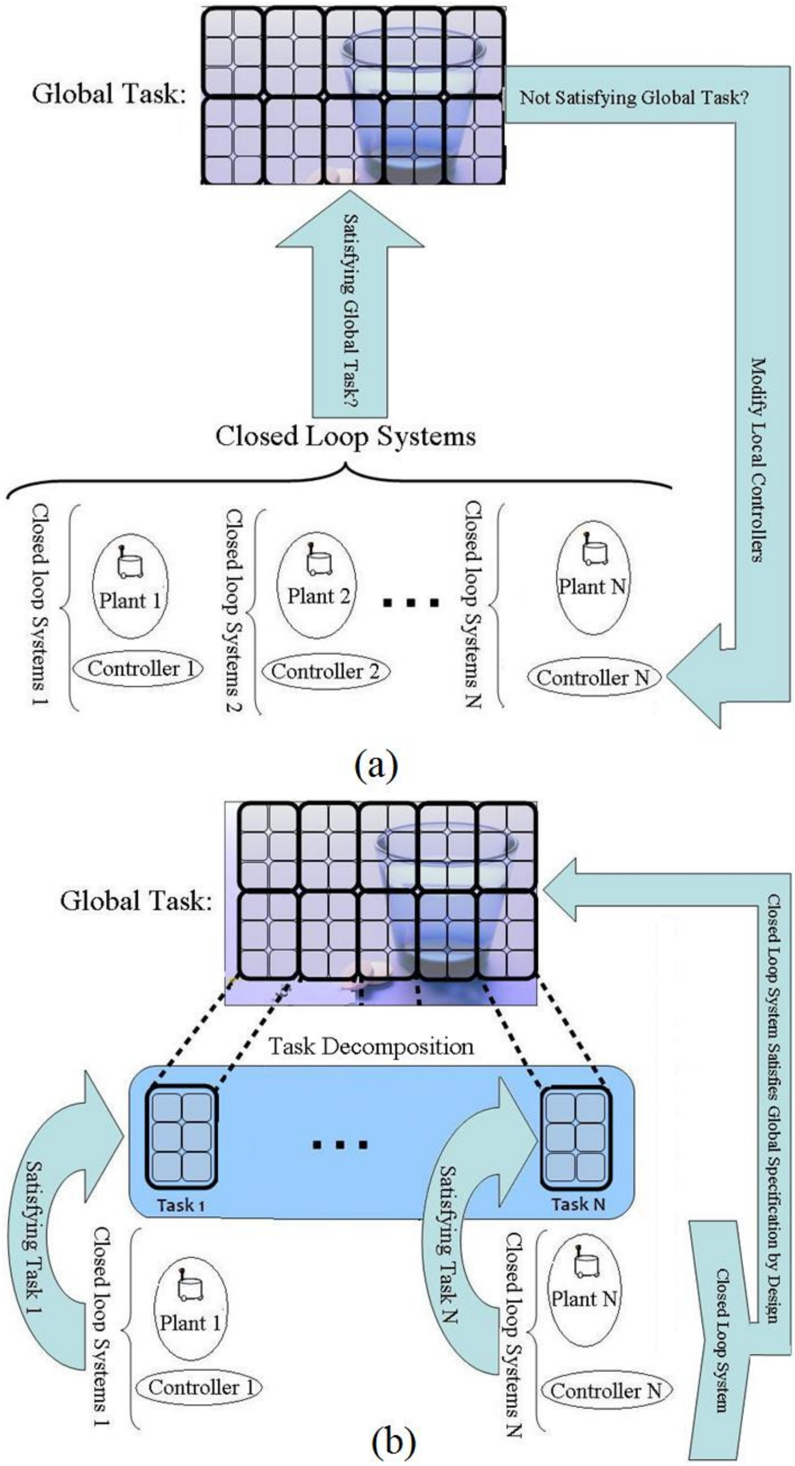


Figure 1.1: (a): Bottom-Up approach, (b): Top-Down approach.

representation of physical event transitions and can express ordering and selections between the events in the global tasks.

1.2 Specifications, Logical Behaviors and Automata Models

Cooperative control of multi-agent systems in general concerns with two types of dynamics to be controlled: continuous-state and discrete event dynamics. While continuous-state systems are time-driven, represented by differential/difference equations and modeled by transfer functions and state space equations; a discrete event system (DES) is event-driven, represented by the sequence of states/events and modeled by sequential models such as languages, automata and Petri nets. A continuous system is controlled in the low-level (inner-loop) of the hierarchy to track an exact trajectory and meet specifications such as stability, optimality and performance. A discrete even system, on the other hand, is controlled (supervised) in a high level (outer-loop) perspective to visit a desired sequence of regions in a partitioned state space, to meet logical specifications. A discrete event system, therefore, can be seen as an event-driven system whose state transitions are triggered by instantaneous events in a discrete state space. In general, continuous controllers deal with control signals based on set-points and control policies, provided by discrete supervisors. In this sense, a discrete event system can represent an abstraction of a continuous/hybrid system and facilitates the evaluation of symbolic (logical) behavior of the system without a detailed investigation of its time-driven quantitative dynamics. This thesis works on the cooperative control of a multi-agent system to achieve logical global specifications.

The first step to control the logical behavior of a team of agents is to represent the desired logical behavior of the team in a mathematical way to capture event-driven transitions between the states of each agent as well as the interactions among agents that allow synchronization over cooperative actions. This perspective is very close to human understanding from the high level behavior of the system that is encoded in sequential flowcharts, in which each part has its own sequence while the parts may synchronize on some events for cooperative actions. In this case, the global specification can be defined over the union of local event sets since the global transitions are defined either individually or synchronously over the events from sensor and actuator signals of the agents. For example, consider two robot agents that do a cooperative surveillance task and are supposed to react upon the first detection of an object, such that “if Agent 1 recognized the object” or “Agent 2 recognized the object”, then for example they synchronously push a door. In this case, “Agent 1 recognized the object” and “Agent 2 recognized the object” are considered as private events, while “pushing the door” is interpreted as a shared event to synchronize on. The global event set is then considered as the union of local event sets and each agent has its own local events to transit individually, while shares some events with its neighbors to synchronize on cooperative actions. In this sense, each local agent perceives the global task from the perspective of its local events. Following example shows a global specification defined over the union of two local event sets for two sub-plants.

Example 1.1 Consider two sequential belt conveyors feeding a bin, as depicted in Figure 1.2. To avoid the overaccumulation of materials on Belt B, when the bin needs to be charged, at first Belt B and then (after a few seconds), Belt A should be started. After filling the bin, to stop the charge, first Belt A and then after a few seconds Belt B is stopped to get completely emptied. The global task, showing the order of events in this plant, is shown in Figure 1.3. The local event sets for Belt A and Belt B are $E_A = \{A_{Start}, Bin_{Full}, A_{Stop}\}$ and $E_B = \{B_{Start}, B_{Stop}, Bin_{Empty}\}$, respectively, with A_{Start}:= Belt A start; Bin_{Full}:= Bin full; A_{Stop}:= Belt A stop and wait for 10 Seconds; B_{Start}:= Belt B start and wait for 10 Seconds; B_{Stop}:= Belt B stop, and Bin_{Empty}: Bin empty.

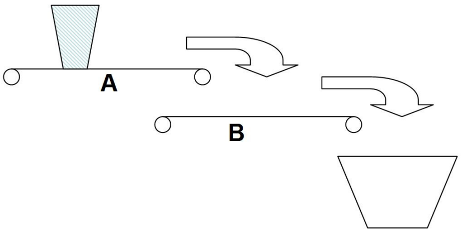


Figure 1.2: The process of two belt conveyors charging a bin.

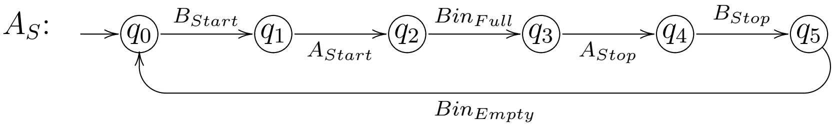


Figure 1.3: Global task automaton for the belt conveyors and bin.

This structure of states and event transitions between the states is called transition system [44] that carries the source state, target state and event or action for each transition.

Definition 1.1 (Transition System [44]) A transition system over a set E of events is a tuple $TS = (S, T, \alpha, \beta, \lambda)$ where S is a set of states; T is a set of transitions; α, β: $T \to S$ denote respectively the *source state* and the *target state* of a transition; $\lambda : T \to E$ denotes the action responsible for the transition, and the mapping $(\alpha, \lambda, \beta) : T \to S \times E \times S$ is one-to-one so that T is a subset of $S \times E \times S$.

A special form of transition system is finite state machine (automaton) with the emphasize on states and traditions, with the following definitions and notations [45].

Definition 1.2 (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 $\delta : Q \times E \to Q$, such that $(q, e, q') \in \delta$ if and only if state q is transited to state q' by event e, denoted by $q \overset{e}{\to} q'$ (or $\delta(q, e) = q'$). A nondeterministic automaton is a tuple $A := (Q, q_0, E, \delta)$ with a partial transition map $\delta : Q \times E \to 2^Q$, and if hidden transitions (ε-moves) are also possible, then a nondeterministic automaton with hidden moves is defined as $A := (Q, q_0, E \cup \{\varepsilon\}, \delta)$ with a partial map $\delta : Q \times (E \cup \{\varepsilon\}) \to 2^Q$. For a nondeterministic automaton, the initial state can be generally from a set $Q_0 \subseteq Q$. 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 (like the following two examples), all states can be considered as marked states and Q_m is omitted from the tuple.

With an abuse of notation, the definitions of transition relation can be extended from the domain of $Q \times E$ (or $Q \times (E \cup \{\varepsilon\})$ in the case of hidden moves) into the domain of $Q \times E^*$ (or $Q \times (E^* \cup \{\varepsilon\})$ for the case of hidden moves) 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 the elements of E).

Definition 1.3 (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 $e \in E$.

Next, for a nondeterministic automaton A (possibly with hidden moves), the ε-closure of $q \in Q$, denoted by $\varepsilon_A^*(q) \subseteq Q$, is recursively defined as: $q \in \varepsilon_A^*(q)$; $q' \in \varepsilon_A^*(q) \Rightarrow \delta(q', \varepsilon) \subseteq \varepsilon_A^*(q)$, based on which $\delta : Q \times E^* \to 2^Q$, is inductively defined as: $\forall q \in Q, s \in E^*, e \in E$: $\delta(q, \varepsilon) := \varepsilon_A^*(q)$ and $\delta(q, se) = \varepsilon_A^*(\delta(\delta(q, s), e)) = \bigcup\limits_{q' \in \delta(q,s)} \left\{ \bigcup\limits_{q'' \in \delta(q',e)} \varepsilon_A^*(q'') \right\}$.

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 $\forall s \in L : \delta(q, s)!$.

For $e \in E$, $s \in E^*$, $e \in s$ means that $\exists t_1, t_2 \in E^*$ such that $s = t_1 e t_2$. In this sense, the intersection of two strings $s_1, s_2 \in E^*$ is defined as $s_1 \cap s_2 = \{e | e \in s_1 \wedge e \in s_2\}$. Likewise, $s_1 \backslash s_2$ is defined as $s_1 \backslash s_2 = \{e | e \in s_1 \wedge e \notin s_2\}$. For $s_1, s_2 \in E^*$, s_1 is called a sub-string of s_2, denoted by $s_1 \leqslant s_2$, when $\exists t \in E^*$, $s_2 = s_1 t$.

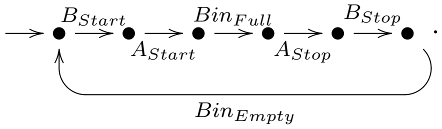
Two events e_1 and e_2 are called successive events if $\exists q \in Q : \delta(q, e_1)! \wedge \delta(\delta(q, e_1), e_2)!$ or $\delta(q, e_2)! \wedge \delta(\delta(q, e_2), e_1)!$. Two events e_1 and e_2 are called adjacent events if $\exists q \in Q : \delta(q, e_1)! \wedge \delta(q, e_2)!$.

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

Definition 1.4 The operator $Ac(.)$ [46] is 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 $Q_{ac} = \{q \in Q | \exists s \in E^*, q \in \delta(q_0, s)\}$ and $\delta_{ac} = \delta | Q_{ac} \times E \to Q_{ac}$, restricting δ to the smaller domain of Q_{ac}. Since $Ac(.)$ has no effect on the behavior of the automaton, from now on we take $A = Ac(A)$.

Following two examples show global task automata defined over the union of local even sets for the team of agents.

Example 1.2 Consider the plant of two sequential belt conveyors and storage bin, in Example 1.1. The task automaton A_S is defined as $A_S = (Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}, q_0, E = \{A_{Start}, Bin_{Full}, A_{Stop}, B_{Start}, B_{Stop}, Bin_{Empty}\}, \delta)$, where $E = E_A \cup E_B$ with respective local event sets $E_A = \{A_{Start}, Bin_{Full}, A_{Stop}\}$ and $E_B = \{B_{Start}, B_{Stop}, Bin_{Empty}\}$. Transition relation on this task automaton is defined as $\delta(q_0, B_{Start}) = q_1$, $\delta(q_1, A_{Start}) = q_2$, $\delta(q_2, Bin_{Full}) = q_3$, $\delta(q_3, A_{Stop}) = q_4$, $\delta(q_4, B_{Stop}) = q_5$, $\delta(q_5, Bin_{Empty}) = q_0$.

When the states are clear from the context, such as this example, the task automaton can be shown with unlabeled states as A_S: 

.

Example 1.2 showed a task automaton for two sub-plants (two belt conveyors) with satanic positions. In some application local plants refer to mobile agents in which some events represent the change in physical position. Following example shows a global specification for a team of three robot-agents defined over the union of local event sets for agents.

Example 1.3 Consider a cooperative multi-robot system (MRS) configured in Figure 1.4. The MRS consists of three robots R_1, R_2 and R_3. All robots have the same communication and positioning capabilities. Furthermore, the robot R_2 has the rescue and fire-fighting capabilities, while R_1 and R_3 are normal robots with the pushing capability. Initially, all of them are positioned in Room 1. Rooms 2 and 3 are accessible from Room 1 by one-way door D_2 and two-way doors D_1 and D_3, as shown in Figure 1.4. All doors are equipped with spring to be closed automatically,

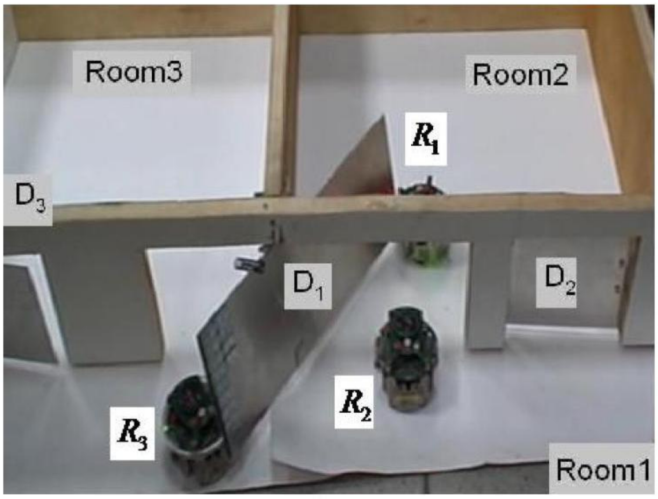


Figure 1.4: The environment of the MRS coordination example.

when there is no force to keep them open.

A_S:

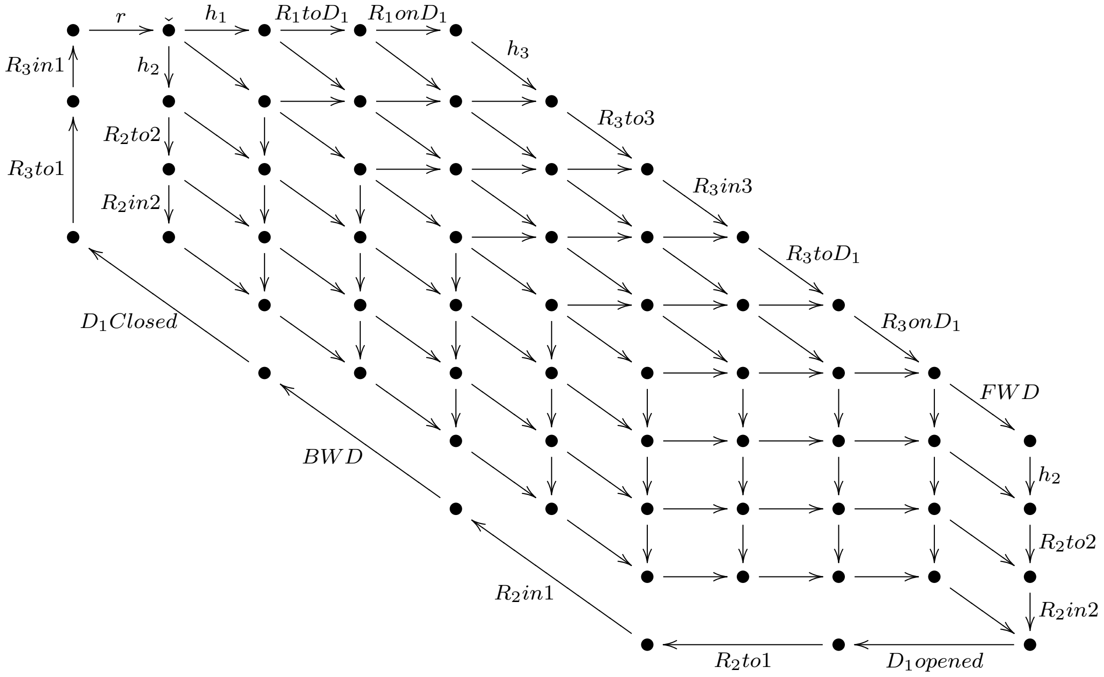


Figure 1.5: Task automaton A_S for the robot team.

Assume that Room 2 requests help for fire extinguishing. After the help announcement, the Robot R_2 is required to go to Room 2, urgently from D_2 and accomplish its task there and come back immediately to Room 1. However, D_2 is a one-way door, and D_1 is a heavy door and 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 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 and then R_3 returns back to Room 1 from D_3. All robots then stay at Room 1 for the next task.

These requirements can be translated into a task automaton for the robot team as it is illustrated in Figure 1.5, 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_iink:= R_i has gone to Room k, $i = 1, 2, 3$, $k = 1, 2, 3$.

As it was illustrated in previous examples, the automata models are user friendly, graphical, easy to visualize and carry the direct physical meaning of events and transitions between the states. It is also reported [47] that among common models for logical behaviors [48–56], automata have a moderate level of abstraction, have reasonable expressive power (as they can represent the human like sequential transitions, can represent the propositional properties [57] and are connected to temporal logic [12, 17, 58–64]) as well as verification power and span a wide range in relation to the life cycle of modeling of DES [47]. It should be noted that the automata models, like other modeling methods, require human experts to define the events based on sensor readings and actuator signals and to translate the requirements into the transitions between the states. Moreover, the representation of global behavior could be complicated as it was shown in Example 1.3; however, automata models are suitable for implementation since they can be realized by simple "if-then" rules over the transitions. This property is particularly important in decentralized cooperative control that local tasks and local controller automata are reduced in the structure defined over the corresponding local event sets. Another advantage of automaton is that, the synthesis of supervisor for a given plant automaton is supported by the product and parallel compositions as well as by mapping on the generated languages. In addition, as it will be discussed in the following section, the behavior of the closed loop system and the specification automata can be compared by equivalence relations [65–67], and moreover, parallel composition and natural projection can be utilized to facilitate the decentralized and modular synthesis and analysis of distributed DES [42, 52, 68].

We focus on deterministic global task automata that are simpler to be characterized, and cover a wide class of specifications such as language specifications that have been well studied in the context of supervisory control of discrete event systems [52]. The advantage of deterministic automaton is that they can uniquely encode the sequence of events using the notion of state and transition relation, and hence do not require huge memory to save the strings as the history of transitions. The qualitative behavior of a deterministic system is therefore described by the set of all possible sequences of events starting from the initial state. Each such a sequence is called a string as described in Definition 1.3, and the collection of all strings represents the language generated by the automaton, denoted by $L(A)$, as defined as follows.

Definition 1.5 (Language, Language Equivalent Automata) For a given automaton A with event set E, the language generated by A is defined as $L(A) := \{s \in E^* | \delta(q_0, s)!\}$. Two automata A_1 and A_2 are said to be language equivalent if $L(A_1) = L(A_2)$.

So far, global task automaton has been discussed for the representation of desired logical behavior of the team of agents. Once the global specification is given for the team, the key problem to accomplish the top-down cooperative tasking idea is the task decomposition such that the composition of local tasks resembles (be equivalent to) the original task. Consequently, several issues have to be declared here: how to obtain local tasks from the global task? how to compose them to retrieve the global task? and how to compare the composition of local tasks with the global task? (what equivalent relation has to be used for the comparison?), as will be discussed in the following three sections, respectively. Next section discusses natural projections to obtain the local task automata as perceived observation of each agent from the global task automaton.

1.3 Natural Projection and Local Task Automata

Given a global task automaton, the cooperative tasking relies on task automaton decomposition such that local task automata collectively resemble the original task automaton. The first step in this top-down approach is therefore to obtain the local task automata. Since each agent has its private events (corresponding to local sensors and actuators) as well as shared events (those sensors and actuator signals that are shared to synchronize on), the global task automaton is defined over the union of all agents' event sets. Consequently, each agent has a local observation from the global task: the perceived global task, filtered by its local event set, i.e., through natural projections with respect to each agent's event set. Accordingly, each local task automaton is the local observer automaton [46], as the interpretation of each agent from the global task. Namely, the agent will ignore the transitions marked by the events that are not in its own event set, i.e., blinds to these moves. The obtained automaton will be a sub-automaton of the global task automaton by deleting all the moves triggered by the blind events of the agent. Particularly, natural projection is defined on automata as $P_{E_i} : A \to A$, where, A is the set of finite automata and $P_{E_i}(A_S)$ are obtained from A_S by replacing its events that belong to $E \backslash E_i$ by ε-moves, and then, merging the ε-related states. The ε-related states form equivalent classes defined as follows.

Definition 1.6 (Equivalent Class of States, [69]) 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 $Q_{/\sim_{E'}} = \{[q]_{E'} | q \in Q\}$.

$\sim_{E'}$ 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 $p_{E'} : E^* \to E'^*$, 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

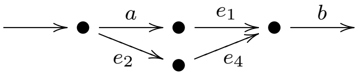
Definition 1.7 (Natural Projection on String, [46]) Consider a global event set E and an event set $E' \subseteq E$. Then, the natural projection $p_{E'} : E^* \to E'^*$ is inductively defined as $p_{E'}(\varepsilon) = \varepsilon$, and $\forall s \in E^*, e \in E : p_{E'}(se) = \begin{cases} p_{E'}(s)e & \text{if } e \in E'; \\ p_{E'}(s) & \text{otherwise.} \end{cases}$

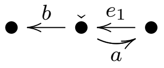
Accordingly, inverse natural projection $p_{E'}^{-1} : E'^* \to 2^{E^*}$ is defined on an string $t \in E'^*$ as $p_{E'}^{-1}(t) := \{s \in E^* | p_{E'}(s) = t\}$. When it is clear from the context, p_i is used instead of p_{E_i}, for simplicity.

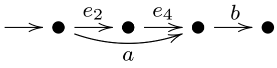
The natural projection is then formally defined on an automaton as follows.

Definition 1.8 (Natural Projection on Automaton) Consider an automaton $A_S = (Q, q_0, E, \delta)$ and an event set $E' \subseteq E$. Then, $P_{E'}(A_S) = (Q_{/\sim_{E'}}, [q_0]_{E'}, E', \delta')$, 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 $q_1' \in \delta(q_1, e)$. 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.

Following example elaborates the concept of natural projection on a given automaton.

Example 1.4 Consider an automaton A_S: 

with the event set $E = E_1 \cup E_2$ and local event sets $E_1 = \{a, b, e_1\}$, $E_2 = \{a, b, e_2, e_4\}$. The natural projection of A_S into E_1 is obtained as $P_1(A_S)$: 

by replacing $e_2, e_4 \in E \backslash E_1$ with ε and merging the ε-related states. Similarly, the projection $P_2(A_S)$ is obtained as $P_2(A_S)$: 

.

Given a task automaton and its local event sets, it is always feasible to do the projection operation, but the question is whether the obtained sub-automata preserve the required specifications in the sense that the fulfillment of each agent with its corresponding sub-task automaton will imply the satisfaction of the global specification as a group.

In order to perform the decomposition, after obtaining local task automata, we need to declare that how to compose local automata and how to compare the collective task automaton (the composition of local task automata) with the global task automaton, as will be discussed in the following two sections.

1.4 Composition of Automata

In order for the correctness of task automaton decomposition, the composition of local automata should resemble the global task automaton, for which it requires to carry the synchronization information to capture the interactions between the agents.

For this purpose, we consider synchronization information to be inserted in each local automaton (instead of considering a coordinating automaton [70, 71]), while no new synchronization events [72] are allowed. In this case, the size of global event set will not be enlarged by inserting unobservable events. This method also allows decentralized synthesis without requiring a centralized coordinator for synchronization. Moreover, we consider predefined local event sets with private events for individual moves as well as shared events for synchronization with neighbors. So far, several versions of operators have been introduced for such synchronization method, namely for composition of local automata. One way to compose local automata and synchronize them on their shared events is vector synchronous product [73]. This operator restricts two systems to simultaneously evolve in each step: a shared event can be transited if it is defined in and transited from the current states of both systems. The private event, on the other hand, can individually evolve in the vector form. In this case, if one of the systems has no available private event, it evolves with an unobservable event, called "stay" or "wait" event. A more liberal operator is state-dependent vector synchronous product that defines the synchronization based on the available shared events in the current states, in the sense that different events in two systems can form a vector transition as long as none of the events are shared events available in current states of both systems [73]. In the state-independent vector synchronous product, however, two different events can form a vector event only if none of them is a shared event. Another vector-based composition is multi-agent product that requires two systems have a transition at every step; and an identical event has to be necessarily synchronized when is available in the current states of both systems [73]. Finally, the most relaxed vector-based composition is simultaneous product [73] that restricts two automata to have a transition at every state with no other requirements.

Another composition of automata is introduced as synchronously communicating systems [74] that is partially vector-based (on common events) and partially scalar (on private events). This composition is based on Zeilonka's asynchronous automata [75] and makes local common transitions conditional to their counterpart transitions in other local automata, such that common transitions joint before forming the global transitions. This means that each branch of a nondeterministic transition in one automaton can be corresponded to only one of the nondeterministic transitions in the other automaton. Although, theoretically, synchronously communicating systems are more expressive, it is practically difficult to distinguish target states after nondeterministic transitions.

In general, although vector-based compositions keep the information of local transitions in vector events, the standard scalar parallel composition is more tractable as the composed automaton can be treated as one system with a scalar language. In this case, the orders, selections and logical properties can be easier investigated and compared with the desired specifications. The supervisory synthesis also is well-established and easier to implement for scalar automata. We therefore adopt the well-accepted operator of parallel composition of automata.

Parallel composition in [46] (loosely cooperating systems in [74], mixed product in [76], synchronous product in [45]), as an special case of synchronized product in [44], captures the synchronization of automata and allows the composed system to be a scalar automaton, such that the individual local transitions can independently evolve, while the transitions on any shared event can evolve globally only if it is transited from the current state of all those local automata that recognize that event. Especial cases of parallel composition are product composition (or completely synchronous compositions) that allows only global transitions on common events; and shuffle product that is defined for automata with mutual exclusive event sets, and only captures the interleaving of private events with no synchronization. Parallel composition itself, however captures the individual private moves and synchronized common transitions in a unified framework.

Parallel composition is therefore a very simple and tractable tool for the modeling of both individual moves as well as synchronized transitions among interactive plants. Accordingly, the parallel composition can be used to capture the collective perception of the team of agents from the global task automaton, i.e., to obtain the collective task automaton by composing the local task automata. Parallel composition is also used to model each local closed loop system by composing its local plant and local controller automata.

Definition 1.9 (Parallel Composition)

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 $A_1||A_2 = (Q = Q_1 \times Q_2, q_0 = (q_1^0, q_2^0), E = E_1 \cup E_2, \delta)$, with δ defined as $\forall (q_1, q_2) \in Q$, $e \in E$:

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

The parallel composition of A_i, $i = 1, 2, ..., n$ is called parallel distributed system

[74] (or concurrent system [69]), and is defined based on the associativity property of parallel composition [46] as $\overset{n}{\underset{i=1}{\|}} A_i = A_1 \parallel \ ... \parallel \ A_n = A_n \parallel (A_{n-1} \parallel (\cdots \parallel (A_2 \parallel A_1)))$.

The set of labels of local event sets containing an event e is called the set of locations of e, denoted by $loc(e)$ and is defined as $loc(e) = \{i \in \{1, \ldots, n\} | e \in E_i\}$. An event e is then called a shared event if $|loc(e)| > 1$. Here, the operator $|.|$ denotes the cardinality or the number of elements of the set.

To investigate the interactions of transitions in two automata, particularly in $P_i(A_S)$, $i = 1, \ldots, n$, the synchronized product of languages is defined as follows.

Definition 1.10 (Synchronized Product of Languages [77]) Consider a global event set E and local event sets E_i, $i = 1, \ldots, n$, such that $E = \overset{n}{\underset{i=1}{\cup}} E_i$. For a finite set of languages $\{L_i \subseteq E_i^*\}_{i=1}^n$, the synchronized product (product language) of $\{L_i\}$, denoted by $\overset{n}{\underset{i=1}{|}} L_i$, is defined as $\overset{n}{\underset{i=1}{|}} L_i = \{s \in E^* | \forall i \in \{1, \ldots, n\} : p_i(s) \in L_i\} = \overset{n}{\underset{i=1}{\cap}} p_i^{-1}(L_i)$.

Using the product language, it is then possible to characterize the language of parallel composition of two automata in terms of their languages as

Lemma 1.1 *([77]) Consider two automata A_1 and A_2, with respective event sets E_1 and E_2. Then, $L(A_1||A_2) = L(A_1)|L(A_2) = p_1^{-1}(L(A_1)) \cap p_2^{-1}(L(A_2))$ with $p_i : E_1 \cup E_2 \to E_i$, $i = 1, 2$.*

Corollary 1.1 *(Interleaving of two strings) Let $A_1 = (\{q_1, ..., q_n\}, \{q_1\}, E_1 = \{e_1, ..., e_n\}, \delta_1)$ and $A_2 = (\{q'_1, ..., q'_m\}, \{q'_1\}, E_2 = \{e'_1, ..., e'_m\}, \delta_2)$ denote path automata (automata with only one branch) $q_1 \overset{e_1}{\to} q_2 \overset{e_2}{\to} \ldots \overset{e_n}{\to} q_n$ and $q'_1 \overset{e'_1}{\to} q'_2 \overset{e'_2}{\to}$*

... $\overset{e'_m}{\to} q'_m$, respectively. Then, $L(A_1||A_2) = \bar{s}|\bar{s'} = p_1^{-1}(\bar{s}) \cap p_2^{-1}(\bar{s'})$ with $s = e_1, ..., e_n$, $s' = e'_1, ..., e'_m$ and $p_i : E_1 \cup E_2 \to E_i$, $i = 1, 2$.

Example 1.5 Consider three strings $s_1 = e_1a$, $s_2 = ae_2$ and $s_3 = ae_1$. Then, the interleaving of s_1 and s_2 is $\overline{s_1}|\overline{s_2} = \overline{e_1ae_2}$, while the interleaving of two strings s_2 and s_3 becomes $\overline{s_2}|\overline{s_3} = \overline{\{ae_1e_2, ae_2e_1\}}$.

Adopting the natural projection to obtain the local task automata, and parallel composition to define the interactions between the local automata, the next issue would be the comparison of automata in order to ensure the correctness of the decomposition. Particularly, it should be declared that what equivalence relation is considered for the comparison of the collective task and global task automata, as it is discussed in the next section.

1.5 Comparison of Automata

To compare the logical behavior of two automata, particularly the collective and global task automata, generally three main equivalence relations are considered in the literature: state-space isomorphism; language equivalence, and bisimulation. Accordingly, the decomposability of task automaton with respect to parallel composition and these equivalence relations are called synthesis modulo language equivalence, synthesis modulo isomorphism, and synthesis modulo bisimulation [74].

The weakest equivalence relation is the language equivalence by which two automata A_1 and A_2 are said to be language equivalent if $L(A_1) = L(A_2)$. Language equivalence can capture sequence, selection and order between the transitions, while bisimulation carries more temporal and branching properties [65]. In particular, bisimulation allows to compare the sequential behavior of two automata using states, that requires less memory rather than using language equivalence. In this case instead of storing and comparison of strings in two automata, one just needs to compare the post-transition of the corresponding states in two automata. In general, bisimilarity implies language equivalence but the converse does not necessarily hold [65]. If two automata A_1 and A_2 are both deterministic, then their bisimulation is reduced to language equivalence. For our application, on the other hand, we will show that (see Example 1.8) even for a deterministic global task automaton A_S, the collective desired behavior $\overset{n}{\underset{i=1}{\|}} P_i\left(A_S\right)$ can be nondeterministic, and hence to compare their behaviors, using the post-transitions of the corresponding states, we need the bisimulation between A_S and $\overset{n}{\underset{i=1}{\|}} P_i\left(A_S\right)$.

Definition 1.11 (Bisimulation [46]) Consider two automata $A_i = (Q_i, q_i^0,\ E, \delta_i)$, $i = 1, 2$. A simulation relation from A_1 to A_2 over $Q_1' \subseteq Q_1$, $Q_2' \subseteq Q_2$ and with respect to $E' \subseteq E$ is a binary relation $R \subseteq Q_1' \times Q_2'$, where

1. $\forall q_1 \in Q_1'$, $\exists q_2 \in Q_2'$ such that $(q_1, q_2) \in R$;
2. if $(q_1, q_2) \in R$, $e \in E'$, $q_1' \in \delta_1(q_1, e)$, then $\exists q_2' \in Q_2'$ such that $\exists q_2' \in \delta_2(q_2, e)$, $(q_1', q_2') \in R$.

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 $q_2' \in \delta_2(q_2, e)$, $(q_1', q_2') \in R$ [46].

If $A_1 \prec A_2$ with a simulation relation R, $A_2 \prec A_1$ with the simulation relation R and R is a symmetric relation (i.e., $\forall (x, y) \in R \Rightarrow (y, x) \in R$ or $R^{-1} = R$), then A_1 and A_2 are said to be bisimilar (bisimulate each other), denoted by $A_1 \cong A_2$ [78]. Equivalently, $A_1 \cong A_2$ when $A_1 \prec A_2$ with a simulation relation R_1, $A_2 \prec A_1$ with a simulation relation R_2 and $R_1^{-1} = R_2$ [79].

It should be noted here that a stronger equivalence relation is isomorphism that has been used in the literature for automaton decomposition, and is formally defined as follows.

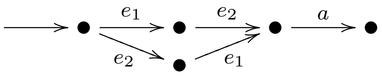
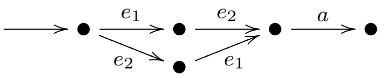
Definition 1.12 (Isomorphism, [80]) Two automata $A_i = (Q_i, q_i^0, E, \delta_i)$, $i = 1, 2$, are said to be isomorphic, denoted by $A_1 \equiv A_2$ if there exists an isomorphism θ from A_1 to A_2 defined as a bijective function $\theta : Q_1 \to Q_2$ such that $\theta(q_1^0) = q_2^0$, and $\theta(\delta_1(q, e)) = \delta_2(\theta(q), e)$, $\forall q \in Q_1, e \in E$.

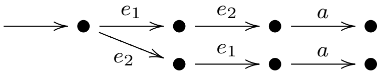
By this definition, two isomorphic automata are bisimilar, but bisimilar automata are not necessarily isomorphic.

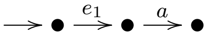
The synthesis modulo isomorphism is more restrictive (as it requires two automata to have the same structure of states and transitions, such that the only allowed difference is the label of states), but easier to be characterized, as it has been presented in [74]. Such strong equivalence relation has been applied in graph theory and computer science synthesis problems [69]. For control applications, on the other hand, other two relations, i.e., language equivalence and bisimulation, are expressive enough and more applicable to capture the behaviors. Language equivalence can compare the behavior of two systems in terms of the sequences of events. For cooperative tasking application, however, since the collective task ($\overset{n}{\underset{i=1}{\|}} P_i(A_S)$) might be nondeterministic, then the bisimulation equivalence is used for the comparison of A_S and $\overset{n}{\underset{i=1}{\|}} P_i(A_S)$; hence, this work focuses on automaton decomposability in the sense of bisimulation as formally stated as follows. From now on, the term "decomposability" refers to the "decomposability in the sense of bisimulation", unless it is stated.

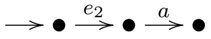
Definition 1.13 (Automaton Decomposability) A task automaton A_S with the event set E and local event sets E_i, $i = 1, ..., n$, $E = \overset{n}{\underset{i=1}{\cup}} E_i$, is said to be decomposable with respect to parallel composition and natural projections P_i, $i = 1, \cdots, n$, in the sense of bisimulation, or in short decomposable, if $\overset{n}{\underset{i=1}{\|}} P_i(A_S) \cong A_S$.

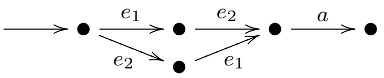
Now, let us see several motivating examples to illustrate the automaton decomposability.

Example 1.6 This example shows an automaton that is decomposable in the sense of all three equivalence relations, isomorphism, bisimulation and languages equivalence. Consider the task automaton A_S:  with local event sets $E_1 = \{e_1, a\}$ and $E_2 = \{e_2, a\}$. For this automaton, $P_1(A_S)$: $\longrightarrow \bullet \xrightarrow{e_1} \bullet \xrightarrow{a} \bullet$, $P_2(A_S)$: $\longrightarrow \bullet \xrightarrow{e_2} \bullet \xrightarrow{a} \bullet$, $P_1(A_S)||P_2(A_S)$: ; hence, $A_S \equiv P_1(A_S)||P_2(A_S)$, $A_S \cong P_1(A_S)||P_2(A_S)$ and $L(A_S) = L(P_1(A_S)||P_2(A_S))$.

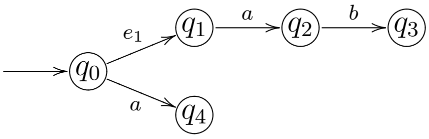
Example 1.7 This example shows an automaton that is decomposable in the sense of bisimulation and language equivalence relations, but not isomorphism. Consider the task automaton A_S: 

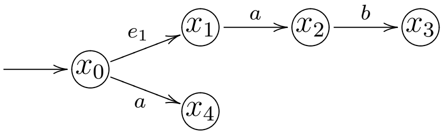
with local event sets $E_1 = \{e_1, a\}$ and $E_2 = \{e_2, a\}$. In this case, $P_1(A_S)$: 

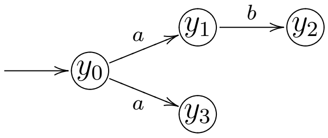
, $P_2(A_S)$: 

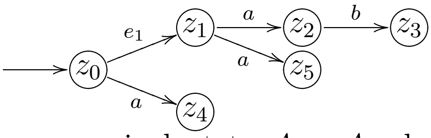
, $P_1(A_S)||P_2(A_S)$: 

; hence, $A_S \cong P_1(A_S)||P_2(A_S)$ and $L(A_S) = L(P_1(A_S)||P_2(A_S))$, but $A_S \not\equiv P_1(A_S)||P_2(A_S)$.

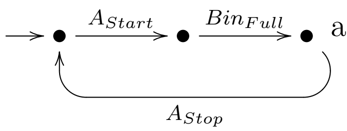
Example 1.8 This example shows an automaton that is decomposable in the sense of only language equivalence but not bisimulation nor isomorphism. Consider A_S: 

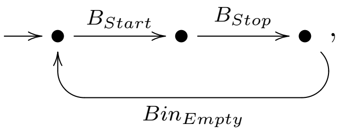
with $E_1 = \{a, b, e_1\}$, $E_2 = \{a, b\}$, leading to $P_1(A_S)$: 

, $P_2(A_S)$: 

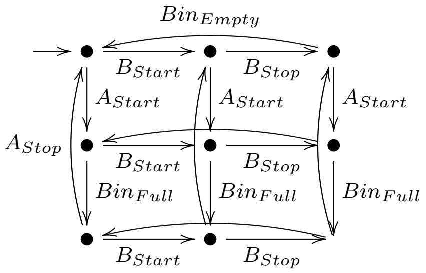
, and $P_1(A_S)||P_2(A_S)$: 

which is neither bisimilar not isomorphic, but language equivalent to A_S. A_S does not bisimulate $P_1(A_S)||P_2(A_S)$, since $\delta(q_0, e_1a) = q_2$ in A_S, $z_5 \in \delta_{||}(z_0, e_1a)$ in $P_1(A_S)||P_2(A_S)$, $\delta(q_2, b)!$ but $\neg\delta_{||}(z_5, b)!$. Moreover, A_S and $P_1(A_S)||P_2(A_S)$ are not isomorphic as there is no state in A_S that forms a bijective pair with z_5. However, A_S has the same language with $P_1(A_S)||P_2(A_S)$ as $L(A_S) = P_1(A_S)||P_2(A_S) = \{\varepsilon, a, e_1, e_1a, e_1ab\}$.

Example 1.9 This example shows an automaton that is not decomposable in the sense of language equivalence and hence is not decomposable in the sense of bisimulation and isomorphism. Consider the task automaton in Example 1.2 with local task automata $P_1(A_S)$: 

and $P_2(A_S)$: 

,

and collective task automaton $P_1(A_S)||P_2(A_S)$: 

that is neither language equivalent, nor bisimilar, nor isomorphic to A_S.

Based on the definition of automaton decomposability and motivating examples we are now ready to state the underlying problems to be tackled in this thesis.

1.6 Problems to be Tackled in the Thesis

While, Examples 1.6 and 1.7 show decomposable automata, Examples 1.8 and 1.9 illustrate instances of task automata that are not decomposable, implying that not all automata are decomposable with respect to parallel composition and natural projections. Then, a natural follow-up question is what makes an automaton decomposable.

Once the task automaton decomposability is characterized by necessary and sufficient conditions, the main question will arise to understand that for a decomposable task automaton whether the fulfilment of local task automata implies the satisfaction of the global task automaton, collectively.

After identifying the conditions for cooperative tasking, next interesting question would be the robustness of the proposed method to understand that if after the design, some events fail in some agents then whether the global task still can be collectively achieved by the team of agents. To answer this question we also need to understand the conditions for task decomposability under event failure. In particular, whether a decomposable task automaton remains decomposable under event failures, and if not what are the necessary and sufficient conditions for preserving the decomposability, in spite of failures.

Example 1.10 Consider the global task automaton in Example 1.3. The task automaton is decomposable with respect to parallel composition and local event sets E_i, $i = 1, 2, 3$, as the parallel composition of local task automata $P_1(A_S)$, $P_2(A_S)$ and $P_3(A_S)$, shown in Figure 1.6, bisimulates A_S.

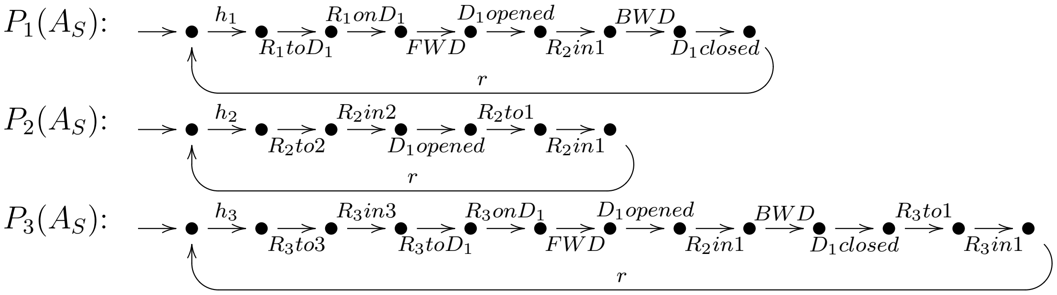


Figure 1.6: $P_1(A_S)$ for R_1; $P_2(A_S)$ for R_2 and $P_3(A_S)$ for R_3.

In this scenario, R_1 and R_3 receive R_2in1 from R_2; R_1 and R_2 receive r and $D_1closed$ from R_3; R_2 and R_3 receive $D_1opened$ from R_1, and R_1 and R_3 share FWD and BWD to synchronize on. Now, an important question is that which of these events are crucial to be shared in order to preserve the decomposability of A_S. In other words, which of these communication links can be safely failed among the agents and which of them has to be sustained to maintain the task decomposability. This example will be revisited after the formulation of event failure in Example 4.3 in Chapter 4, to illustrate the notion of fault-tolerant task decomposability.

Next interesting and more practically important question would be the investigation of task automata that are not decomposable. In this case, we would like to ask whether it is possible to make them decomposable and if so, whether the automata can be made decomposable with the minimum number of communication links.

Example 1.11 Consider the task automaton in Example 1.2 for the process of belt conveyors and storage bin. The task automaton is not decomposable as it was shown in Example 1.9. The question is now whether it is possible to decompose such task automaton and if yes, whether it is possible to do it with the minimum number of communication links (This example will be revisited in Example 5.1 in Chapter 5 to elaborate the enforcing of the task decomposability).

To answer these questions, the objectives of the thesis are outlined as follows:

1. to understand that given a deterministic task automaton and a set of local event sets, whether the task automaton is always decomposable, such that the parallel composition of local task automata is equivalent (in the sense of bisimulation) to the global task automaton and if not, what are the necessary and sufficient conditions for such decomposability?
2. if the task automaton is decomposable and assuming the existence of local supervisors to enforce local task automata, is it guaranteed that the entire closed loop system satisfies (bisimulates) the global task automaton?
3. if the task automaton is decomposable and local supervisors exist to enforce local tasks and the global specification is satisfied, but some events fail in some agents, then whether the global task automaton remains decomposable and if

not, what are the necessary and sufficient conditions for preserving the task decomposability in spite of event failures?, and

4. if the decomposability conditions under event failures are satisfied, whether the global specification can be still collectively satisfied by the team of agents, in spite of event failures, and

5. if the task automaton is not decomposable, is it possible to make it decomposable by modifying the distribution of events among the agents?

1.7 Organization of the Thesis

To address the top-down cooperative control, three fundamental questions are evoked here: The first question is the task decomposition problem that is interested in understanding of whether all tasks are decomposable and if not, what are the conditions for task decomposability. It furthermore asks that if the task is decomposable and local controllers exist to satisfy local tasks, whether the whole closed loop system satisfies the global specification. Subsequently, the second question refers to the cooperative control under event failures, and would like to know if after the task decomposition and local controller designs for global satisfaction, some events fail in some agents, then whether the task still remains decomposable and globally satisfied, in spite of event failures. As another interesting direction, the third question investigates a way to make an indecomposable task decomposable through the modification of local agents in order to accomplish the proposed cooperative control.

For the cooperative control of logical behaviors [81], represented in automata

[45, 46], the first question (task decomposability for cooperative tasking) is addressed in Chapter 2, by decomposing a given global task automaton into two local task automata such that their parallel composition bisimulates the original task automaton. By using the notion of shared events, instead of common events and incorporating the concept of global decision making on the orders and selections between the transitions, the decomposability result is generalized in Chapter 3 into an arbitrary finite number of agents. Given a deterministic task automaton, and a set of local event sets, necessary and sufficient conditions are identified for task automaton decomposability based on decision making on the orders and selections of transitions, the interleaving of synchronized strings and the determinism of bisimulation quotient of local automata. It is also proven that the fulfillment of local task automata guarantees the satisfaction of the global specification, by design. A cooperative control scenario has been developed and implemented for a team of three robots and the results are shown to illustrate the concept of task decomposition and cooperative tasking.

The second question, cooperative tasking under event failures, is investigated in Chapter 4, by introducing a notion of passive events to transform the fault-tolerant task decomposability problem into the standard automaton decomposability in Chapters 2 and 3. The passivity is found to reflect the redundancy of communication links, based on which the necessary and sufficient conditions are introduced under which a previously decomposable task automaton remains decomposable, in spite of event failures. The conditions ensure that after passive failures, the team of agents maintains its capability for global decision making on the orders and selections between transitions; no illegal behavior is allowed by the team (no new string emerges in the interleavings of local strings) and no legal behavior is disabled by the team (no string in the global task automaton is stopped in the parallel composition of local automata). These conditions interestingly guarantee the team of agents to still satisfy its global specification, even if some local agents fail to maintain their local specifications.

The third question is addressed in Chapter 5 to investigate what is exactly needed to make an originally indecomposable task automaton decomposable, and how to make it decomposable. For a global task automaton that is not decomposable with respect to given local event sets, the problem is particularly interested in finding a way to modify the local task automata such that their parallel composition bisimulates the original global task automaton, to guarantee its satisfaction by fulfilling the local task automata. For this purpose, a procedure is proposed to firstly use the results in Chapter 4 to remove the redundant communication links (that are passive and their deletions respect the decomposability conditions under event failure), and then identify the sources of indecomposability, using the results in Chapters 2, 3. Any indecomposability in this stage is not due to redundant links; hence, it cannot be overcome by link deletion. The algorithm will instead suggest establishing new links to enforce each decomposability condition. Finding the sequence of link additions to enforce the task automaton decomposability is found to require an exhaustive dynamic search. Therefore, besides a graph theory approach to enforce the global decision making on orders and selections, simple sufficient conditions are introduced to avoid illegal interleavings and local nondeterminisms and also to prevent checking of the corresponding conditions after each link addition. This method can make any indecomposable task automaton decomposable in order for cooperative control of multi-agent systems.

The thesis is finally concluded in Chapter 6 with highlighting the contributions and outlining the future works.

Chapter 2

Cooperative Tasking for Two Agents

2.1 Introduction

This chapter aims to propose a task decomposition approach applicable in divide-and-conquer synthesis for cooperative multi-agent systems, as it was motivated in the very beginning. The decomposition should be in such a way that the fulfilment of local tasks collectively lead to the satisfaction of the global specification. The main underlying question here is to understand whether it is always possible to decompose a given task automaton and if not, what are the necessary and sufficient conditions for its decomposability.

To formally describe the specification, a deterministic finite automaton is chosen here to represent the global specifications for multi-agent systems, due to its expressibility for a large class of tasks [45,46], its similarity to our human logical commands, and its connection to the temporal logic specifications [61,82]. Accordingly, we will focus on the logical behavior of a multi-agent system and model its collective behavior through parallel composition [74]. It is assumed that the global task automaton is defined over the union of all agents' event sets and hence, local task automata are obtained through natural projections with respect to each agent's event set. Given a task automaton and the sets of local events, it is always feasible to do the projection operation, but the question is whether the obtained sub-automata preserve the required specifications in the sense that the fulfillment of each agent with its corresponding sub-task automaton will imply the satisfaction of the global specification as a group. Unfortunately, by a simple counterexample, it can be shown that the answer is not always (see Examples 1.8 and 1.9). Then, a natural follow-up question is what the necessary and sufficient conditions should be for the proposed decomposability of a global specification. The main part of the chapter is set to answer this question. The automaton decomposability problem is particularly important in the area of distributed supervisory control of discrete event systems.

The field of supervisory control of discrete event systems has been formally initiated by Ramadge and Wonham [52, 83] to control the logical behavior of systems, pertaining evolutions according to the asynchronous occurrence of events. It has been also extended to supervisory control under partial observation [84], supervisory control of vector discrete event systems [85, 86] and concurrent vectors discrete event systems [87], supervisory control of multi-agent product systems [88–90] (based on multi-agent product [73]) and bisimilarity control [91].

For multi-agent systems with high complexity [92–95] due to many interacting subsystems, and large-scale distribution of sensors and actuators, it is more efficient, flexible and scalable to synthesis distributed supervisory control [96, 97]. In distributed supervisory control, several supervisors cooperatively deal with smaller sub-plants or/and sub-specifications, in modular and decentralized configurations. In modular supervisory control, the supervisor is comprised of several sub-supervisors each of which for a module of plant or/and specification. On the other hand, decentralized supervisory control refers to the structure with several "local" supervisors (not necessarily corresponding to the modules of plants or/and specifications), each of which with local sensing and actuating capabilities, that jointly lead the controlled closed loop system to satisfy a global specification. Decentralized approaches also increase the robustness, flexibility and functionality of the systems due to the parallelism of local agents, with some degree of redundancy. These approaches also offer more mobility and autonomy to agents due to local sensing and actuation.

The examples of distributed supervisory control include modular centralized supervisory control with several cooperative supervisors with access to all events [98–101]; non-modular decentralized supervisory control, with several supervisors with partial accesses to the events of a monolithic plant [102–110]; modular decentralized supervisory control with indecomposable task, where local supervisors correspond to modules of plants but a global specification [111–116], and modular decentralized supervisory control with decomposable task, where both plant and specification are decomposable and each local supervisor corresponds to one module of plant and specification [74, 77, 91, 103, 117–120]. Among all of these configurations, decentralized modular structures simplify the synthesis and implementation of top-down cooperative control. In these structures the plant is given as a parallel distributed system (parallel composition of local plant automata) and the global specification is represented as a decomposable task automaton and each local controller is designed for the corresponding modules of local plant and local specification automata. Therefore, the key problem to accomplish this top-down idea is task decomposition such that the composition of local tasks resembles (be equivalent to) the original task.

Similar automaton decomposition problem has been studied in the computer science literature. Roughly speaking, two different classes of problems have been studied, so far. The first problem is to design the event distribution so as to make the automaton decomposable, which is typically studied in the context of concurrent systems. For example, [69] characterized the conditions for decomposition of asynchronous automata in the sense of isomorphism based on the maximal cliques of the dependency graph. The isomorphism equivalence used in [69] is however a strong condition, in the sense that two isomorphic automata are bisimilar but not vise versa [46]. In many applications bisimulation relation suffices to capture the equivalence relationship. On the other hand, the second class of problems assumes that the distribution of the global event set is given and the objective is to find conditions on the automaton such that it is decomposable. This is usually called synthesis modulo problem [74] that can be investigated under three types of equivalence: isomorphism, bisimulation and language equivalence, as it was discussed in Chapter 1. Bisimulation synthesis modulo for a global automaton has been also addressed in [74, 121], by introducing necessary and sufficient conditions for automaton decomposability based on the product language of the automaton and the determinism of its bisimulation quotient. Obtaining the bisimulation quotient, however, is generally a difficult task, and the condition on product language relies on language separability [77], which is indeed another form of decomposability. These problems motivate us to develop new necessary and sufficient conditions that can characterize the decomposability based on the investigation of events and strings in the given automaton.

In this chapter, we identify conditions on the global specification automaton in terms of its private and common events for the proposed decomposability, which are shown to be necessary and sufficient for the case of two agents. In the next chapter, the result will be generalized into the case of arbitrary finite number of agents. Furthermore, it will be shown that if the global task is decomposable, then designing the local controller for each agent to satisfy its corresponding sub-task will lead the entire multi-agent system to achieve the global specification.

The rest of this chapter is organized as follows. Section 2.2 introduces the necessary and sufficient conditions for the decomposability of an automaton with respect to parallel composition and two local event sets. This section builds the foundation for the next chapter for generalizing the result into an arbitrary finite number of agents. The chapter then concludes with remarks and discussions in Section 2.3. The proofs of lemmas are given in the Appendix.

2.2 Task Decomposability for Two Agents

As it was motivated in the Chapter 1, the essential issue in cooperative tasking is task automaton decomposition that is formally stated as follows.

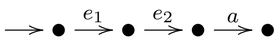
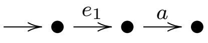
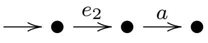
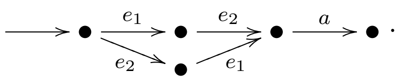
Problem 2.1 *(Task Decomposability Problem) Given a deterministic task automaton A_S with the event set $E = \overset{n}{\underset{i=1}{\cup}} E_i$ and the local event sets E_i, $i = 1, \ldots, n$, what are the necessary and sufficient conditions that A_S is decomposable with respect to parallel* *composition and natural projections P_i, $i = 1, \cdots, n$, such that $\overset{n}{\underset{i=1}{\|}} P_i\,(A_S) \cong A_S$?*

This chapter answers Problem 2.1 by introducing necessary and sufficient conditions for task automaton decomposability for two cooperative agents ($n = 2$). Next chapter will then generalize this result for an arbitrary finite number of agents and furthermore addresses the more important question of cooperative tasking to understand that whether a decomposable global task automaton is guaranteed upon the satisfaction of local specifications.

In order for $A_S \cong P_1(A_S)||P_2(A_S)$, from the definition of bisimulation, it is required to have $A_S \prec P_1(A_S)||P_2(A_S)$ (with a relation R_1); $P_1(A_S)||P_2(A_S) \prec A_S$ (with a relation R_2), and $R_1^{-1} = R_2$. These requirements are provided by the following three lemmas. Firstly, the following simulation relationship always holds true.

Lemma 2.1 *Consider any deterministic automaton A_S with event set $E = E_1 \cup E_2$, local event sets E_i, and natural projections P_i, $i = 1, 2$. Then, $A_S \prec P_1(A_S)||P_2(A_S)$.*

Proof: See the proof in the Appendix. ∎

Example 2.1 Consider an automaton A_S to be  with local event sets $E_1 = \{e_1, a\}$ and $E_2 = \{e_2, a\}$. The parallel composition of $P_1(A_S)$:  and $P_2(A_S)$:  is $P_1(A_S)||P_2(A_S)$:  One can observe that $A_S \prec P_1(A_S)||P_2(A_S)$ but $P_1(A_S)||P_2(A_S) \not\prec A_S$, meaning that A_S is not decomposable with respect to parallel composition and natural projections P_i, $i = 1, 2$.

Lemma 2.1 says that, in general, $P_1(A_S)||P_2(A_S)$ simulates A_S. The similarity of $P_1(A_S)||P_2(A_S)$ to A_S, however, is not always true (see Example 2.1), and needs some conditions as stated in the following lemma.

Lemma 2.2 *Consider a deterministic automaton* $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$ *and natural projections* P_i*,* $i = 1, 2$*. Then,* $P_1(A_S)||P_2(A_S) \prec A_S$ *if and only if* A_S *satisfies the following conditions:* $\forall e_1 \in E_1 \backslash E_2, e_2 \in E_2 \backslash E_1, q \in Q,\ s \in E^*$*:*

- $DC1 : [\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*
- $DC2 : \delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!$*, and*
- $DC3 : \forall s, s' \in E^*,\ s \neq s',\ p_{E_i \cap E_j}(s),\ p_{E_i \cap E_j}(s')$ *start with the same common event* $a \in E_1 \cap E_2$*,* $q \in Q$*:* $\delta(q, s)! \wedge \delta(q, s')! \Rightarrow \delta(q, \overline{p_1(s)}|\overline{p_2(s')})! \wedge \delta(q, \overline{p_1(s')}|\overline{p_2(s)})!$.

Proof: See the proof in the Appendix. ∎

Next, we need to show that for two simulation relations R_1 (for $A_S \prec P_1(A_S)||P_2(A_S)$) and R_2 (for $P_1(A_S)||P_2(A_S) \prec A_S$), defined by Lemmas 2.1 and 2.2, $R_1^{-1} = R_2$.

Lemma 2.3 *Consider an automaton* $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$ *with natural projections* P_i*,* $i = 1, 2$ *and* $P_1(A_S)||P_2(A_S) = (Z, z_0, E, \delta_{||})$*. If* A_S *is deterministic,* $A_S \prec P_1(A_S)||P_2(A_S)$ *with the simulation relation* R_1 *and* $P_1(A_S)||P_2(A_S) \prec A_S$ *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:* $\forall i \in \{1, 2\},\ x, x_1, x_2 \in Q_i,\ x_1 \neq x_2,\ e \in E_i,\ t \in E_i^*,\ x_1 \in \delta_i(x, e),\ x_2 \in \delta_i(x, e)$*:* $\delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$.

Proof: See the proof in the Appendix. ■

Based on these lemmas, the main result on task automaton decomposition is given as follows.

Theorem 2.1 *(Task Decomposability for Two Agents) A deterministic automaton* $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$ *is decomposable with respect to parallel composition and natural projections* P_i, $i = 1, 2$, *such that* $A_S \cong P_1(A_S)||P_2(A_S)$ *if and only if* A_S *satisfies the following decomposability conditions (DC):* $\forall e_1 \in E_1 \backslash E_2, e_2 \in E_2 \backslash E_1, q \in Q$, $s \in E^*$,

- $DC1$*:* $[\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*
- $DC2$*:* $\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!$*, and*
- $DC3$ *:* $\forall s, s' \in E^*$, $s \neq s'$, $p_{E_i \cap E_j}(s)$, $p_{E_i \cap E_j}(s')$ *start with the same common event* $a \in E_1 \cap E_2$, $q \in Q$*:* $\delta(q, s)! \wedge \delta(q, s')! \Rightarrow \delta(q, \overline{p_1(s)}|\overline{p_2(s')})! \wedge \delta(q, \overline{p_1(s')}|\overline{p_2(s)})!$*;*
- $DC4$*:* $\forall i \in \{1, 2\}$, $x, x_1, x_2 \in Q_i$, $x_1 \neq x_2$, $e \in E_i$, $t \in E_i^*$, $x_1 \in \delta_i(x, e)$, $x_2 \in \delta_i(x, e)$*:* $\delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$.

Proof: According to Definition 1.11, $A_S \cong P_1(A_S)||P_2(A_S)$ if and only if $A_S \prec P_1(A_S)||P_2(A_S)$ (that is always true due to Lemma 2.1), $P_1(A_S)||P_2(A_S) \prec A_S$ (that it is true if and only if $DC1$, $DC2$ and $DC3$ are true, according to Lemma 2.2) and one of the simulation relations is the inverse of the other relation, i.e., $R_1^{-1} = R_2$ (that for a deterministic automaton A_S, when $A_S \prec P_1(A_S)||P_2(A_S)$ with the simulation relation R_1 and $P_1(A_S)||P_2(A_S) \prec A_S$ with the simulation relation R_2, due to Lemma 2.3, $R_1^{-1} = R_2$ holds true if and only if $DC4$ is satisfied). Therefore, $A_S \cong P_1(A_S)||P_2(A_S)$

if and only if $DC1$, $DC2$, $DC3$ and $DC4$ are satisfied. ∎

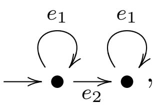
Remark 2.1 Intuitively, the decomposability condition $DC1$ means that for any adjacent pair (decision on switch) of private events $(e_1, e_2) \in \{(E_1 \backslash E_2, E_2 \backslash E_1), (E_2 \backslash E_1, E_1 \backslash E_2)\}$ (from different private event sets), both orders e_1e_2 and e_2e_1 should be legal from the same state. $DC2$ states that for any pair of successive private events from different event sets (decision on order), before any string, they should be allowed to occur in any order. It should be noted that here, e_1e_2 and e_2e_1 are not required to meet at the same state (unlike FD and ID in [69] for decomposability in the sense of isomorphism); but due to $DC2$, any string $s \in E^*$ after them should be the same, or in other words, if e_1 and e_2 are necessary conditions for the occurrence of a string s, then any order of these two events would be legal for such occurrence (see Example 2.1). Note that, as a special case, s could be ε. This condition is similar to the notion of trace [121], that intuitively carries the concept of independent successive events that can occurs in any order (A language $L \in E^*$ is said to be a trace language if $\forall e_1, e_2 \in E, \forall \omega, \omega' \in E^* : \omega e_1 e_2 \omega' \in L$ and $\nexists E_i \in \{E_1, \dots, E_n\}$, $e_1, e_2 \subseteq E_i$, then $\omega e_2 e_1 \omega' \in L$). The notion of trace, however cannot capture the condition on decision making on adjacent events.

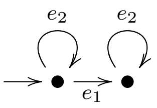
The condition $DC3$ means that if two strings s and s' share the same first appearing common event, then any interleaving of these two strings should be legal in A_S. This requirement is due to the synchronization of projections of these strings in $P_1(A_S)$ and $P_2(A_S)$. The last condition, $DC4$, ensures the symmetry of mutual simulation relations between A_S and $P_1(A_S)||P_2(A_S)$. Given the determinism of A_S, this symmetry is guaranteed when each local task automaton bisimulates a deterministic automaton, leading to the existence of a deterministic automaton that is bisimilar to $P_1(A_S)||P_2(A_S)$. If $R_1^{-1} \neq R_2$, then some of the sequences that are allowed in A_S will be disabled in $P_1(A_S)||P_2(A_S)$.

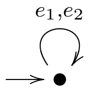
A related notion to our work is language decomposability [112, 113] that is a type of decentralized observability [122, 123], and is defined as $K = L \cap p_1^{-1}(p_1(K)) \cap p_2^{-1}(p_2(K))$ for language plant L, specification language K and natural projections p_1, p_2, and means that agents have enough information to retrieve the global specification, i.e., "any string in K can be distinguished by at least one of the projections p_1 or p_2, otherwise any retrieval should fall in K" [112]. This in turn means that any order of any successive events in any string of K should be legal, or at least one the projections $p_1(K)$ or $p_2(K)$ should be capable of distinguishing this order. The language decomposability also implicitly embodies $DC1$ and $DC3$, stating that the global languages specification should contain all possible interleaving languages of all local languages. The condition $DC4$, on the other hand, is relaxed for the language decomposability as the languages do not capture the nondeterminism; hence, the mutual set inclusion for two languages implies their equality; unlike the mutual simulation of two automata that does not imply their bisimilarity, in general. This is identically hold true for the weaker notion of called language separability $K = p_1^{-1}(p_1(K)) \cap p_2^{-1}(p_2(K))$ [77]. Automaton decomposability in the sense of bisimulation, on the other hand, besides the checking the capability of local plants on decision making on the switches ($DC1$) and orders ($DC2$) of events, and that the synchronization of local task automata should not lead to an illegal behavior($DC3$); it also requires local task automata to present deterministic behaviors ($DC4$), in order to allow all legal strings of A_S to appear in $P_1(A_S)||P_2(A_S)$.

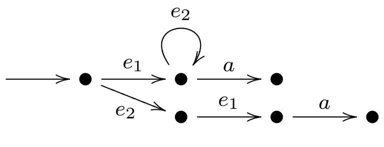
The decomposability conditions can be then paraphrased as follows: Any decision on order or switch between two transitions that cannot be made locally (by at least one local controller) should not be critical globally (any result of the decision should be allowed); and the interpretation of the global task by the team of local plants should neither allow an illegal behavior (a string that is not in global task automaton), nor disallow a legal behavior (a string that appears in the global task automaton).

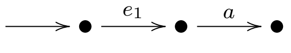
Another point is that in general, the task automaton may contain loops that introduce moves that are successive/adjacent to other transitions. The decomposability of such automata also can be checked using $DC1$-$DC4$. Examples of decomposable automata with self-loops are shown in the following example.

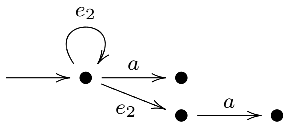
Example 2.2 Consider the automata A_1: 

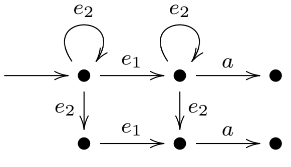
, A_2: 

and A_3: 

with $E_1 = \{e_1\}, E_2 = \{e_2\}$. All of these automata satisfy $DC1$ and $DC2$ as $\delta(q_0, e_1e_2)! \wedge \delta(q_0, e_2e_1)!$. They also satisfy $DC3$ since there is no common event shared between E_1 and E_2. Finally, $DC4$ is also fulfilled for them since there is no nondeterminism in their local automata. The automaton A_4: 

with $E_1 = \{a, e_1\}$, $E_2 = \{a, e_2\}$ is not decomposable as the parallel compositions of $P_1(A_4) \cong$ 

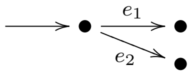
and $P_2(A_4)$: 

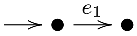
is $P_1(A_4)||P_2(A_4) \cong$ 

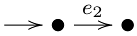
$\ncong A_4$. The

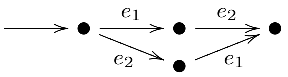
automaton A_4 is not decomposable since it violates $DC2$, as $\delta(q_0, e_1e_2e_2)!$ but $\neg\delta(q_0, e_2e_1e_2))!$ in A_4.

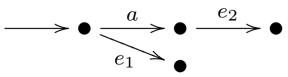
Now, to elaborate the decomposability conditions, following four examples illustrate $DC1$-$DC4$ for decomposable and indecomposable automata.

Example 2.3 This example illustrates the concept of decision making on switching between the events. Furthermore, it shows an automaton that satisfies $DC2$, $DC3$ and $DC4$, but not $DC1$, leading to indecomposability. The automaton A_S: 

with local event sets $E_1 = \{e_1\}$ and $E_2 = \{e_2\}$, is not decomposable as the parallel composition of $P_1(A_S)$: 

and $P_2(A_S)$: 

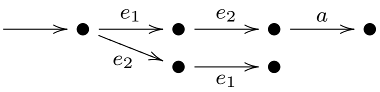
is $P_1(A_S)||P_2(A_S)$: 

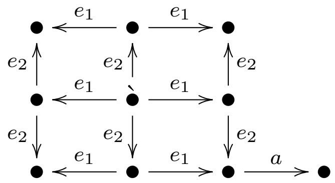
which does not bisimulate A_S. Here, A_S is not decomposable with respect to parallel composition and natural projections P_i, $i = 1, 2$, since two events $e_1 \in E_1 \backslash E_2$ and $e_2 \in E_2 \backslash E_1$ do not respect $DC1$, as none of the local plants take in charge of decision making on the switching between these two events. One can observe that, if in this example $e_1 \in E_1 \backslash E_2$ and $e_2 \in E_2 \backslash E_1$ were separated by a common event $a \in E_1 \cap E_2$, then 

with local event sets $E_1 = \{e_1, a\}$ and $E_2 = \{e_2, a\}$, was decomposable, since the decision on the switch between e_1 and a could be made in E_1 and then E_2 could be responsible for the decision on the order of a and e_2.

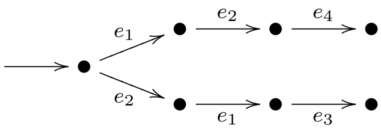
Example 2.4 The automaton A_S in Example 2.1 shows an automaton that respects $DC1$, $DC3$ and $DC4$, but is indecomposable due to the violation of $DC2$. Here, A_S is not decomposable since none of the local plants take in charge of decision making on the order of two events $e_1 \in E_1 \backslash E_2$ and $e_2 \in E_2 \backslash E_1$. The two automata in

Examples 1.6 and 1.7 satisfy $DC2$ as both orders of e_1 and e_2 are allowed, before a. Also, if in Example 2.1 $e_1 \in E_1 \backslash E_2$ and $e_2 \in E_2 \backslash E_1$ were separated by a common event $a \in E_1 \cap E_2$, then the automaton $\to \bullet \xrightarrow{e_1} \bullet \xrightarrow{a} \bullet \xrightarrow{e_2} \bullet$ with local event sets $E_1 = \{e_1, a\}$ and $E_2 = \{e_2, a\}$, was decomposable, since the decision on the orders of e_1 and a and then a and e_2 could be made in E_1 and then E_2, subsequently. As another example, consider an automaton A_S: with $E_1 = \{a, e_1\}$, $E_2 = \{a, e_2\}$. This automaton is not decomposable due to the violation of $DC2$, as $P_1(A_S)||P_2(A_S)$: $\ncong A_S$. The transition $\delta_{||}(z_0, e_2e_1a)!$ in $P_1(A_S)||P_2(A_S)$, but $\neg\delta(q_0, e_2e_1a)!$ in A_S. Therefore, A_S is not decomposable. If the lower branch in A_S was continued with a transition on a after e_2e_1, then the automaton was decomposable (see Example 1.7).

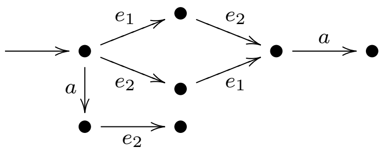


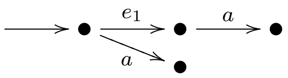


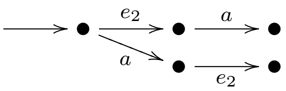
As another example that satisfies $DC1$ and $DC3$ but not $DC2$, consider A_S: with $E_1 = \{e_1, e_3\}$, $E_2 = \{e_2, e_4\}$.

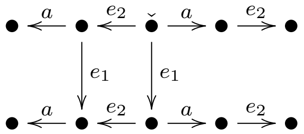


Example 2.5 This example illustrates an automaton that satisfies $DC1$, $DC2$ and $DC4$, but it is indecomposable as it does not fulfill $DC3$, since new strings appear in $P_1(A_S)||P_2(A_S)$ from the interleaving of two strings in $P_1(A_S)$ and $P_2(A_S)$, but they are not legal in A_S. Consider the task automaton A_S: with $E_1 = \{a, e_1\}$, $E_2 = \{a, e_2\}$, lead-



ing to $P_1(A_S) \cong$ 

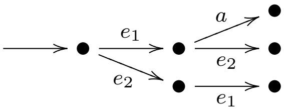
, $P_2(A_S) \cong$ 

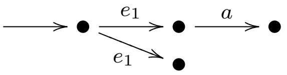
and $P_1(A_S)||P_2(A_S)$: 

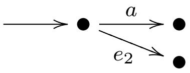
that is not bisimilar to A_S since two strings e_2a and e_1ae_2 are newly generated, while they do not appear in A_S, although both $P_1(A_S)$ and $P_2(A_S)$ are deterministic.

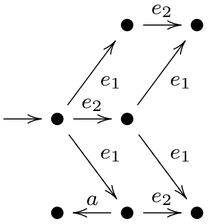
Example 2.6 This example illustrates an automaton that satisfies $DC1$ and $DC2$, and $DC3$, but is indecomposable as it does not fulfill $DC4$. Consider the task automaton A_S in Example 1.8, where $A_S \not\cong P_1(A_S)||P_2(A_S)$ due to the violation of $DC4$. The task automaton A_S satisfies $DC1$ and $DC2$ as contains no successive/adjacent transitions defined on different local event sets. It also satisfies $DC3$ as any string in $T = \{\overline{p_1(s)}|\overline{p_2(s')}, \overline{p_1(s')}|\overline{p_2(s)}\}$ (s and s' are the top and bottom strings in A_S and share the first appearing common event $a \in E_1 \cap E_2$), appears in A_S. But, it does not fulfill $DC4$, since there exists a transition on string e_1a from z_0 to z_5 that stops in $P_1(A_S)||P_2(A_S)$, whereas, although e_1a transits from q_0 in A_S, it does not stop afterwards. This illustrates the dissymmetry in simulation relations between A_S and $P_1(A_S)||P_2(A_S)$. Note that $A_S \prec P_1(A_S)||P_2(A_S)$ with the simulation relation R_1 over all events in E, from all states in Q into some states in Z, as $R_1 = \{(q_0, z_0), (q_1, z_1), (q_2, z_2), (q_3, z_3), (q_4, z_4)\}$. Moreover, $P_1(A_S)||P_2(A_S) \prec A_S$ with the simulation relation R_2 over all events in E, from all states in Z into some states in Q, as $R_2 = \{(z_0, q_0), (z_1, q_1), (z_2, q_2), (z_3, q_3), (z_4, q_4), (z_5, q_2)\}$. Therefore, although $A_S \prec P_1(A_S)||P_2(A_S)$ and $P_1(A_S)||P_2(A_S) \prec A_S$, $P_1(A_S)||P_2(A_S) \not\cong A_S$, since $\exists(z_5, q_2) \in R_2$, whereas $(q_2, z_5) \notin R_1$.

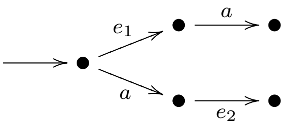
If for stopping of string e_1a in $P_1(A_S)||P_2(A_S)$, there was a state in Q reachable from q_0 by e_1a and stopping there, then we would have $\forall q \in Q, z \in Z : (q, z) \in R_1 \Leftrightarrow (z, q) \in R_2$ and $P_1(A_S)||P_2(A_S) \cong A_S$.

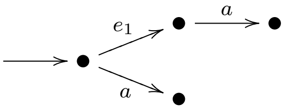
It should be noted that the condition $DC4$ not only applies for nondeterminism on common events, but also it requires any nondeterminism on any private event also to have a bisimilar deterministic counterpart. For example, consider the task automaton A_S: 

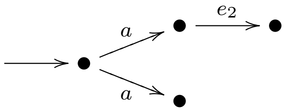
with $E_1 = \{e_1, a\}$, $E_2 = \{e_2, a\}$. The parallel composition of $P_1(A_S)$: 

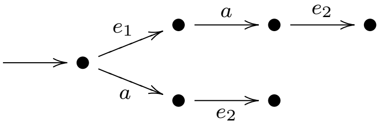
(with nondeterministic transition on private event e_1) and $P_2(A_S) \cong$ 

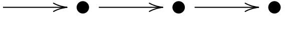
is $P_1(A_S)||P_2(A_S)$: 

which is not bisimilar to A_S. The automaton A_S: 

with $E = E_1 \cup E_2$, $E_1 = \{a, e_1\}$, $E_2 = \{a, e_2\}$, $P_1(A_S)$: 

and $P_2(A_S)$: 

is an example of an indecomposable automaton that violates both $DC3$ and $DC4$. It violates $DC3$ since $\delta_{||}(z_0, e_1ae_2)!$ in $P_1(A_S)||P_2(A_S)$, but $\neg\delta(q_0, e_1ae_2)!$ in A_S, and it does not satisfy $DC4$ since $P_2(A_S)$ is nondeterministic and is not bisimilar to a deterministic automaton, leading to a string in $P_1(A_S)||P_2(A_S)$ that e_2 is disallowed after a while there in no such restriction in A_S. If A_S was A_S: 

, then $P_2(A_S) \cong$ 

, and A_S

was decomposable.

2.3 Conclusion

The chapter proposed a method to check automaton decomposability in order for top-down supervisory control of multi-agent systems. Given a set of two cooperative agents whose logical behaviors can be modeled as a parallel distributed system, and a global task automaton, the chapter has provided necessary and sufficient conditions for decomposability of the task automaton with respect to parallel composition and natural projections into two local event sets.

The result says that a task automaton is decomposable if and only if any pair of adjacent or successive events from different local event sets can be symmetrically transited in any order; and the interleaving of strings from two local task automata neither excludes legal strings of the global task automaton, not exceed from the set of its legal strings.

Next chapter will generalize the proposed approach into an arbitrary finite number of agents, and furthermore will investigate that whether the fulfillment of local tasks implies the satisfaction of a decomposable global task automaton.

2.4 Appendix

2.4.1 Proof for Lemma 2.1

We prove $A_S \prec P_1(A_S)||P_2(A_S))$ by showing that $R = \{(q,z) \in Q \times Z | \exists s \in E^*, \delta(q_0,s) = q, z = ([q]_1,[q]_2)\}$ is a simulation relation, defined on all events in E and all reachable states in A_S. Consider $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$, $P_i(A_S) = (Q_i, q_i^0, E_i, \delta_i)$, $i = 1,2$, $P_1(A_S)||P_2(A_S) = (Z, z_0, E, \delta_{||})$. Then, $\forall q,q' \in Q, e \in E, \delta(q,e) = q'$, according to the definition of natural projection (Definition 1.8) $[q']_i = \begin{cases} \delta_i([q]_i, e) & \text{if } e \in E_i; \\ [q]_i & \text{if } e \notin E_i \end{cases}$, $i = 1,2$, and due to the definition of parallel composition (Definition 1.9) $([q']_1,[q']_2) \in \delta_{||}(([q]_1,[q]_2),e) =$

$$\begin{cases} (\delta_1([q]_1,e), \delta_2([q]_2,e)), & \text{if } e \in E_1 \cap E_2; \\ (\delta_1([q]_1,e), [q]_2), & \text{if } e \in E_1 \backslash E_2; \\ ([q]_1, \delta_2([q]_2,e)), & \text{if } e \in E_2 \backslash E_1. \end{cases}$$

This is true for any $q \in Q$, particularly for q_0. This reasoning can be repeated for any reachable state in Q. Therefore, starting from q_0 and taking $(q_0, z_0 = ([q_0]_1,[q_0]_2)) \in R$, from the above construction, it follows that for any reachable state in Q ($\exists s \in E^*, \delta(q_0,s) = q$) and $q' \in Q$, $e \in E$, $\delta(q,e) = q'$, there exists $z = ([q]_1,[q]_2)$, $z' = ([q']_1,[q']_2)$ such that $z' \in \delta_{||}(z,e)$, and we can take $(q,z) \in R$ and $(q',z') \in R$. Therefore, $R = \{(q,z) \in Q \times Z | \exists s \in E^*, \delta(q_0,s)!, z = ([q]_1,[q]_2)\}$ is a simulation relation, defined over all $e \in E$, and all reachable states in A_S; hence, $A_S \prec P_1(A_S)||P_2(A_S)$.

2.4.2 Proof for Lemma 2.2

We use two following lemmas during the proof.

Lemma 2.4 *Consider a deterministic automaton $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$. Then $DC1 \wedge DC2 \Rightarrow [\forall s \in E^*,\ \delta(q_0, s)! \Rightarrow \delta(q_0, \overline{p_1(s)}|\overline{p_2(s)})!]$ in A_S.*

This lemma means that for any transition defined on a string in A_S, all path automata defined on the interleaving of $\overline{p_1(s)}$ and $\overline{p_2(s)}$ in $P_1(A_S)||P_2(A_S)$ are simulated by A_S, provided $DC1$ and $DC2$.

Proof: Consider a deterministic automaton $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$, a string $s \in E^*$, $\delta(q_0, s) = q$, and its projections $p_1(s)$, $p_2(s)$ with $x \in \delta_1(x_0, \overline{p_1(s)})$, $y \in \delta_2(y_0, \overline{p_2(s)})$ and $(x, y) \in \delta_{||}((x_0, y_0), \overline{p_1(s)}|\overline{p_2(s)})$, in $P_1(A_S)$, $P_2(A_S)$ and $P_1(A_S)||P_2(A_S)$, respectively. Any string s can be written as $s = \omega^1\gamma^1 ...\ \omega^K\gamma^K$, with $\omega^k \in [E \backslash (E_1 \cap E_2)]^*$, $\gamma^k \in (E_1 \cap E_2)^*$, $p_1(\omega^k) = \alpha^k = \alpha_0^k ... \alpha_{m_k}^k$, $\alpha_0^k = \varepsilon$, $p_2(\omega^k) = \beta^k = \beta_0^k ... \beta_{n_k}^k$, $\beta_0^k = \varepsilon$, $p_1(\gamma^k) = p_2(\gamma^k) = \gamma^k = \gamma_0^k ... \gamma_{r_k}^k$, $\gamma_0^k = \varepsilon$. The case $m_k = 0$, $n_k = 0$, $r_k = 0$, $K = 0$, results in $p_1(\omega^k) = \varepsilon$, $p_2(\omega^k) = \varepsilon$, $\gamma^k = \varepsilon$ and $s = \varepsilon$. Based on this setting and the definition of parallel composition, for $k = 0, ..., K$, $i = 0, ..., m_k$, $j = 0, ..., n_k$ and $r = 0, ..., r_k$, the interleaving $\overline{p_1(s)}|\overline{p_2(s)}$ is evolved in $P_1(A_S)||P_2(A_S)$ as follows: $\forall (x_{k+i}, y_{k+j}) \in Q_1 \times Q_2$: $(\delta_1(x_{k+i}, \alpha_i^k), y_{k+j}) \in \delta_{||}((x_{k+i}, y_{k+j}), \alpha_i^k)$, $((\delta_1(x_{k+i}, \alpha_i^k), \delta_2(y_{k+j}, \beta_j^k)) \in \delta_{||}((\delta_1(x_{k+i}, \alpha_i^k), y_{k+j}), \beta_j^k)$, $(x_{k+i}, \delta_2(y_{k+j}, \beta_j^k)) \in \delta_{||}((x_{k+i}, y_{k+j}), \beta_j^k)$, $(\delta_1(x_{k+i}, \alpha_i^k), \delta_2(y_{k+j}, \beta_j^k)) \in \delta_{||}((x_{k+i}, \delta_2(y_{k+j}, \beta_j^k)), \alpha_i^k)$, $(\delta_1(x_{k+m_k+r}, \gamma_r^k), \delta_2(y_{k+n_k+r}, \gamma_r^k)) \in \delta_{||}((x_{k+m_k+r}, y_{k+n_k+r}), \gamma_r^k))$ with $\delta_1(x_{k+i}, \alpha_i^k) \ni \begin{cases} x_{k+i} & \text{if } \alpha_i^k = \varepsilon \\ x_{k+i+1} & \text{if } \alpha_i^k \neq \varepsilon \end{cases}$, $\delta_2(y_{k+j}, \beta_j^k) \ni \begin{cases} y_{k+j} & \text{if } \beta_j^k = \varepsilon \\ y_{k+j+1} & \text{if } \beta_j^k \neq \varepsilon \end{cases}$ and

$$(\delta_1(x_{k+m_k+r}, \gamma_r^k), \delta_2(y_{k+n_k+r}, \gamma_r^k)) \ni \begin{cases} (x_{k+m_k+r}, y_{k+n_k+r}) & \text{if } \gamma_r^k = \varepsilon \\ (x_{k+m_k+r+1}, y_{k+n_k+r+1}) & \text{if } \gamma_r^k \neq \varepsilon \end{cases}.$$

Moreover, $DC1$ and $DC2$ collectively imply that $\forall e_1 \in E_1 \backslash E_2$, $e_2 \in E_2 \backslash E_1$, $q \in Q$, $[\delta(q, e_1)! \wedge \delta(q, e_2)!] \vee \delta(q, e_1e_2)! \vee \delta(q, e_2e_1)! \Rightarrow \delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!$ which particularly means that $\forall k \in \{0, ...K\}$, $\forall \alpha_i^k, \beta_j^k, q_{i,j} \in Q$, $i = 0, ..., m_k$, $j = 0, ..., n_k$: $\delta(q_{k+i,k+j}, \alpha_i^k \beta_j^k)! \wedge \delta(q_{k+i,k+j}, \beta_j^k \alpha_i^k)!$ with a simulation relation $R(\omega^k) = \{((x_{k+i}, y_{k+j}), q_{k+i,k+j}),\ ((\delta_1(x_{k+i}, \alpha_i^k), y_{k+j}),\ \delta(q_{k+i,k+j}, \alpha_i^k)),$ $((x_{k+i}, \delta_2(y_{k+j}, \beta_j^k)), \delta(q_{k+i,k+j}, \beta_j^k)), ((\delta_1(x_{k+i}, \alpha_i^k), \delta_2(y_{k+j}, \beta_j^k)), \delta(q_{k+i,k+j}, \alpha_i^k \beta_j^k)),$ $((\delta_1(x_{k+i}, \alpha_i^k), \delta_2(y_{k+j}, \beta_j^k)), \delta(q_{k+i,k+j}, \beta_j^k \alpha_i^k))\}$ from transitions defined on $\overline{p_1(\omega^k)}|\overline{p_2(\omega^k)}$ into A_S. For the transitions on the common events, the evolutions are $\delta(q_{k+m_k+r,k+n_k+r}, \gamma_r^k)!$ in A_S for $r = 0, ..., r_k$, leading to simulation relation $R(\gamma^k) = \{((x_{k+m_k+r}, y_{k+n_k+r}),\ q_{k+m_k+r,k+n_k+r}),\ ((\delta_1(x_{k+m_k+r}, \gamma_r^k), \delta_2(y_{k+n_k+r}, \gamma_r^k)),$ $\delta(q_{k+m_k+r,k+n_k+r},\ \gamma_r^k))\}$ from transitions on $\overline{p_1(\gamma^k)}|\overline{p_2(\gamma^k)}$ into A_S. Therefore, for $i = 0, ..., m_k$, $j = 0, ..., n_k$, $r = 0, ..., r_k$, $R = \bigcup\limits_{k=0}^{K} R(\omega^k) \cup R(\gamma^k)$ defines a simulation relation from path automata (from z_0 along $\overline{p_1(s)}|\overline{p_2(s)}$) in $P_1(A_S)||P_2(A_S)$ into A_S.

$\blacksquare$

Lemma 2.5 *If $DC1$ and $DC2$ hold true, then $\forall s, s' \in E^*$, $\delta(q_0, s)!$, $\delta(q_0, s')!$, $s \neq s'$, $p_{E_1 \cap E_2}(s)$, $p_{E_1 \cap E_2}(s')$ do not start with the same $a \in E_1 \cap E_2$, then $\delta(q_0, \overline{p_1(s)}|\overline{p_2(s')})! \wedge \delta(q_0, \overline{p_1(s')}|\overline{p_2(s)})!$ in A_S.*

Proof: The antecedent of Lemma 2.5 addresses three following cases: (Case 1): $s = \omega_1$, $s' = \omega_1'$; (Case 2): $s = \omega_1 a \omega_2$, $s' = \omega_1'$, and (Case 3): $s = \omega_1 a \omega_2$, $s' = \omega_1' b \omega_2'$, where, $\omega_1, \omega_1' \in [E \backslash (E_1 \cap E_2)]^*$, $\omega_2, \omega_2' \in (E_1 \cup E_2)^*$, $a, b \in E_1 \cap E_2$.

For Case 1, setting $K = 1$, $r_k = 0$, taking $p_1(\omega_1) = \alpha = \alpha_0 ... \alpha_m \in (E_1 \backslash E_2)^*$, $p_2(\omega'_1) = \beta' = \beta'_0 ... \beta'_{n'} \in (E_2 \backslash E_1)^*$, $p_1(\omega'_1) = \alpha' = \alpha'_0 ... \alpha'_{m'} \in (E_1 \backslash E_2)^*$ and $p_2(\omega_1) = \beta = \beta_0 ... \beta_n \in (E_2 \backslash E_1)^*$, similar to the first part of Lemma 2.4, it follows that $\delta(q_0, \overline{p_1(\omega_1)}|\overline{p_2(\omega'_1)})!$ and $\delta(q_0, \overline{p_1(\omega'_1)}|\overline{p_2(\omega_1)})!$ in A_S.

For Case 2, from Case 1 and Lemma 2.4 it follows that $\delta(q_0, \overline{p_1(s)}|\overline{p_2(s')})! = \delta(q_0, [\overline{p_1(\omega_1)}|\overline{p_2(\omega'_1)}]a\overline{p_1(\omega_2)})!$ and $\delta(q_0, \overline{p_1(s')}|\overline{p_2(s)})! = \delta(q_0, [\overline{p_1(\omega'_1)}|\overline{p_2(\omega_1)}]a\overline{p_2(\omega_2)})!$ in A_S.

For the third case, from the definition of parallel composition combined with the first two cases and also Lemma 2.4, $\delta(q_0, \overline{p_1(s)}|\overline{p_2(s')})!$ leads to $\delta(q_0, [\overline{p_1(\omega_1)}|\overline{p_2(\omega'_1)}]a[\overline{p_1(\omega_2)}|\overline{p_2(\omega_2)}])!$ and $\delta(q_0, [\overline{p_1(\omega_1)}|\overline{p_2(\omega'_1)}]b[\overline{p_1(\omega'_2)}|\overline{p_2(\omega'_2)}])!$ in A_S and similarly, $\delta(q_0, \overline{p_1(s')}|\overline{p_2(s)})!$ results in $\delta(q_0, [\overline{p_1(\omega'_1)}|\overline{p_2(\omega_1)}]a[\overline{p_1(\omega_2)}|\overline{p_2(\omega_2)}])!$ and $\delta(q_0, [\overline{p_1(\omega'_1)}|\overline{p_2(\omega_1)}]b[\overline{p_1(\omega'_2)}|\overline{p_2(\omega'_2)})])!$ in A_S.

Therefore, for all three cases, $\delta(q_0, \overline{p_1(s)}|\overline{p_2(s')})!$ in A_S and $\delta(q_0, \overline{p_1(s')}|\overline{p_2(s)})!$ in A_S. ∎

Now, Lemma 2.2 is proven as follows.

Sufficiency: The set of transitions in $P_1(A_S)||P_2(A_S)$, can be defined as $T = \{(x_0, y_0) \xrightarrow{\overline{p_1(s)}|\overline{p_2(s')}} (x, y) \in Q_1 \times Q_2\}$, where, $(x_0, y_0) \xrightarrow{\overline{p_1(s)}|\overline{p_2(s')}} (x, y)$ in $P_1(A_S)||P_2(A_S)$ is the interleaving of transitions $x_0 \xrightarrow{p_1(s)} x$ in $P_1(A_S)$ and $y_0 \xrightarrow{p_2(s')} y$ in $P_2(A_S)$ (projections of transitions $q_0 \xrightarrow{s} q$ and $q_0 \xrightarrow{s'} q'$, respectively, in A_S). T can be divided into three sets of transitions corresponding to a division $\{\Gamma_1, \Gamma_2, \Gamma_3\}$ on the set of interleaving strings $\Gamma = \{\overline{p_1(s)}|\overline{p_2(s')}|q_0 \xrightarrow{s} q, q_0 \xrightarrow{s'} q', q, q' \in Q, s, s' \in E^*\}$, where, $\Gamma_1 = \{\overline{p_1(s)}|\overline{p_2(s')} \in \Gamma | s = s'\}$, $\Gamma_2 = \{\overline{p_1(s)}|\overline{p_2(s')} \in \Gamma | s \neq s', p_{E_1 \cap E_2}(s)$ and $p_{E_1 \cap E_2}(s')$

do not start with the same event$\}$, and $\Gamma_3 = \{\overline{p_1(s)}|\overline{p_2(s')} \in \Gamma | s \neq s', p_{E_1\cap E_2}(s)$ and $p_{E_1\cap E_2}(s')$ start with the same event $\}$.

Now, provided $DC1$, $DC2$ and $DC3$, for any s, s', $\delta(q_0, s)!$, $\delta(q_0, s')!$, in A_S, both $\delta(q_0, \overline{p_1(s)}|\overline{p_2(s')})!$ and $\delta(q_0, \overline{p_1(s')}|\overline{p_2(s)})!$ are guaranteed, for Γ_1, due to Lemma 2.4; for Γ_2, due to Lemma 2.5, and for Γ_3, due to the combination of $DC3$ and Lemma 2.4 (for simplification, in $DC3$, s and s' can be started from any q, instead of q_0, and the strings between q_0 and q are checked by Lemma 2.4).

Necessity: The necessity is proven by contradiction. Suppose that A_S simulates $P_1(A_S)||P_2(A_S)$, but $\exists e_1 \in E_1\backslash E_2, e_2 \in E_2\backslash E_1, q \in Q, s \in E^*$ s.t. (1): $[\delta(q, e_1)! \wedge \delta(q, e_2)!] \wedge \neg[\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$; (2): $\neg[\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!]$, or (3): $\exists s, s' \in E^*$, sharing the same first appearing common event $a \in E_1 \cap E_2$, $s \neq s'$, $q \in Q$: $\delta(q, s)! \wedge \delta(q, s')! \wedge \neg[\delta(q, \overline{p_1(s)}|\overline{p_2(s')})! \wedge \delta(q, \overline{p_1(s')}|\overline{p_2(s)})!]$.

In the first case, due to the definition of parallel composition, the expression $[\delta(q, e_1)! \wedge \delta(q, e_2)!]$ leads to $\delta_{||}(z, e_1e_2)! = \delta_{||}(z, e_2e_1)!$, where $z \in Q_1 \times Q_2$ in $P_1(A_S)||P_2(A_S)$ corresponds to $q \in Q$ in A_S. Therefore, $\delta_{||}(z, e_1e_2)! \wedge \delta_{||}(z, e_2e_1)!$, but, $\neg[\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$. This means that $P_1(A_S)||P_2(A_S) \not\prec A_S$ which is a contradiction. The second case means $\exists e_1 \in E_1\backslash E_2, e_2 \in E_2\backslash E_1, q \in Q, s \in E^*$ s.t. $[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!] \wedge \neg[\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$. From the definition of parallel composition, then $\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!$ implies that $\delta_{||}(z, e_1e_2)! = \delta_{||}(z, e_2e_1)!$, for some $z \in Q_1 \times Q_2$ corresponding to $q \in Q$. Consequently, from the definition of transition relation and $A_S \prec P_1(A_S)||P_2(A_S)$ it turns to $\delta_{||}(z, e_1e_2s)! = \delta_{||}(z, e_2e_1s)!$, meaning that $\delta_{||}(z, e_1e_2s)! \wedge \delta_{||}(z, e_2e_1s)!$, but, $\neg[\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$. This in turn contradicts with the similarity assumption of $P_1(A_S)||P_2(A_S) \prec A_S$. The third case also leads to the contradiction as it causes the violation of the simulation relation from $P_1(A_S)||P_2(A_S)$ into A_S as $\delta(q, s)! \wedge \delta(q, s')!$ leads to $\delta_{||}(z, \overline{p_1(s)}|\overline{p_2(s')})! \wedge \delta_{||}(z, \overline{p_2(s)}|\overline{p_1(s')})!$ in $P_1(A_S)||P_2(A_S)$, whereas $\neg[\delta(q, \overline{p_1(s)}|\overline{p_2(s')})! \wedge \delta(q, \overline{p_2(s)}|\overline{p_1(s')})!]$.

2.4.3 Proof for Lemma 2.3

Following three lemmas are used during the proof of Lemma 2.3. The first lemma is used for the combination of bisimilarity between automata using parallel compositions.

Lemma 2.6 *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. $(A_1 \cong A_2) \wedge (A_3 \cong A_4) \Rightarrow (A_1 \parallel A_3 \cong A_2 \parallel A_4)$.

Proof: The first part of this lemma is proven by showing that the relation $R = \{((q_1, q_3), (q_2, q_4))|(q_1, q_2) \in R_1 \text{ and } (q_3, q_4) \in R_2\}$ is a simulation relation, where, R_1 and R_2 are the respective simulations from A_1 to A_2 and from A_3 to A_4.

Consider $A_i = (Q_i, q_i^0, E_i, \delta_i)$, $i = 1, ..., 4$, $A_1||A_3 = (Q_{1,3}, (q_1^0, q_3^0), E = E_1 \cup E_3, \delta_{1,3})$, $A_2||A_4 = (Q_{2,4}, (q_2^0, q_4^0), E = E_2 \cup E_4, \delta_{2,4})$, $E_1 = E_3$ and $E_2 = E_4$. Then, $\forall (q_1, q_3), (q_1, q_3)' \in Q_{1,3}$, $e \in E$, $q_2 \in Q_2$, $q_4 \in Q_4$ such that $(q_1, q_3)' \in \delta_{1,3}((q_1, q_3), e)$, $(q_1, q_2) \in R_1$ and $(q_3, q_4) \in R_2$, according to the definition of parallel composition (Definition 1.9), we have $(q_1, q_3)' = (q_1', q_3') =$

$$\begin{cases} (\delta_1(q_1,e),\delta_3(q_3,e))\,, & \text{if } \delta_1(q_1,e)!,\ \delta_3(q_3,e)!,\ e\in E_1\cap E_3; \\ (\delta_1(q_1,e),q_3)\,, & \text{if } \delta_1(q_1,e)!,\ e\in E_1\backslash E_3; \\ (q_1,\delta_3(q_3,e))\,, & \text{if } \delta_3(q_3,e)!,\ e\in E_3\backslash E_1; \end{cases}$$

and due to the definition of simulation (Definition 1.11), $A_1 \prec A_2$ and $A_3 \prec A_4$, it follows that

$$\begin{cases} \exists q_i' \in Q_i, q_i' \in \delta_i(q_i,e), i=2,4 & \text{if } e\in E_2\cap E_4 \\ \exists q_2' \in Q_2, q_2' \in \delta_2(q_2,e) & \text{if } e\in E_2\backslash E_4 \\ \exists q_4' \in Q_4, q_4' \in \delta_4(q_4,e) & \text{if } e\in E_4\backslash E_2 \end{cases}$$

This, in turn, due to the definition of parallel composition implies that $\exists (q_2,q_4)' := (q_2',q_4') \in Q_{2,4}$ such that $(q_2,q_4)' \in \delta_{2,4}((q_2,q_4),e) =$

$$\begin{cases} (\delta_2(q_2,e),\delta_4(q_4,e))\,, & \text{if } e\in E_2\cap E_4; \\ (\delta_2(q_2,e),q_4)\,, & \text{if } e\in E_2\backslash E_4; \\ (q_2,\delta_4(q_4,e))\,, & \text{if } e\in E_4\backslash E_2. \end{cases}$$

Therefore, $\forall (q_1,q_3),(q_1,q_3)' \in Q_{1,3}$, $(q_2,q_4) \in Q_{2,4}$, $e \in E$, such that $(q_1,q_3)' \in \delta_{1,3}((q_1,q_3),e)$ and $((q_1,q_3),(q_2,q_4)) \in R$, then $\exists (q_2,q_4)' \in Q_{2,4}$, $(q_2,q_4)' \in \delta_{2,4}((q_2,q_4),e)$, $((q_1,q_3)',(q_2,q_4)') \in R$. This together with $((q_1^0,q_3^0),(q_2^0,q_4^0)) \in R$, by construction, leads to $A_1||A_3 \prec A_2||A_4$.

Now, to prove the second part of Lemma 2.6, we define the relation $\bar{R} = \{((q_2,q_4),(q_1,q_3))|(q_2,\ q_1) \in \bar{R}_1 \text{ and } (q_4,q_3) \in \bar{R}_2\}$, where, $\bar{R}_1$ and $\bar{R}_2$ are the respective simulation relations from A_2 to A_1 and from A_4 to A_3, and then similar to the proof of the first part, we show that $\bar{R}$ is a simulation relation. Now, to show that $A_1||A_3 \cong A_2||A_4$ it remains to show that $\forall (q_1,q_3) \in Q_{1,3}$, $(q_2,q_4) \in Q_{2,4}$: $((q_1,q_3),(q_2,q_4)) \in R \Leftrightarrow ((q_2,q_4),(q_1,q_3)) \in \bar{R}$. This is proven by contradiction. Suppose that $\exists (q_1,q_3) \in Q_{1,3}$, $(q_2,q_4) \in Q_{2,4}$ such that $((q_1,q_3),(q_2,q_4)) \in$

$R \wedge ((q_2,q_4),(q_1,q_3)) \notin \bar{R}$, or $((q_2,q_4),(q_1,q_3)) \in \bar{R} \wedge ((q_1,q_3),(q_2,q_4)) \notin R$. We prove that the first hypothesis leads to a contradiction, and the contradiction of the second hypothesis is followed, similarly. The expression $((q_1,q_3),(q_2,q_4)) \in R \wedge ((q_2,q_4),(q_1,q_3)) \notin \bar{R}$ means that $\exists s \in E^*$, $(q_1,q_3) \in \delta_{1,3}((q_1^0,q_3^0),s)$, $(q_2,q_4) \in \delta_{2,4}((q_2^0,q_4^0),s)$, $\forall e \in E$, $\delta_{1,3}((q_1,q_3),e)!$: $\delta_{2,4}((q_2,q_4),e)!$; but, $\exists \sigma \in E$, $\delta_{2,4}((q_2,q_4),\sigma)! \wedge \neg\delta_{1,3}((q_1,q_3),\sigma)!$. From Definition 1.9, $\delta_{2,4}((q_2,q_4),\sigma)!$ means that

$$\begin{cases} \delta_2(q_2,\sigma)!, \delta_4(q_4,\sigma)! & \text{if } e \in E_2 \cap E_4; \\ \delta_2(q_2,\sigma)! & \text{if } e \in E_2 \backslash E_4; \\ \delta_4(q_4,\sigma)! & \text{if } e \in E_4 \backslash E_2. \end{cases}$$

Consequently, from $(q_2,q_1) \in \bar{R}_1$ and $E_1 = E_2$ (due to $A_1 \cong A_2$), $(q_4,q_3) \in \bar{R}_2$ and $E_3 = E_4$ (due to $A_3 \cong A_4$), and Definition 1.11, it follows that

$$\begin{cases} \delta_1(q_1,\sigma)!, \delta_3(q_3,\sigma)! & \text{if } e \in E_2 \cap E_4 = E_1 \cap E_3; \\ \delta_1(q_1,\sigma)! & \text{if } e \in E_2 \backslash E_4 = E_1 \backslash E_3; \\ \delta_3(q_3,\sigma)! & \text{if } e \in E_4 \backslash E_2 = E_3 \backslash E_1. \end{cases}$$

, that from Definition 1.9 leads to $\delta_{1,3}((q_1,q_3),\sigma)!$ which contradicts with the hypothesis and the proof is followed. ■

Next, the following lemma is introduced to characterize the symmetric property between simulation relations.

Lemma 2.7 *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 $A_1' \cong A_2$.*

Proof:

Sufficiency: $A_1 \prec A_2$, $A_2 \prec A_1$ and $A_1' \cong A_2$, collectively, result in $A_1 \prec A_1'$ and $A_1' \prec A_1$, that due to the determinism of A_1 and A_1' lead to $A_1 \cong A_1'$. Finally, since $A_1' \cong A_2$, from the transitivity of bisimulation, $A_1 \cong A_2$, and consequently, $R_1^{-1} = R_2$.

Necessity: The necessity is proven by contradiction as follows. Consider two automata $A_1 = (X, x_0, E, \delta_1)$, $A_2 = (Y, y_0, E, \delta_2)$, let A_1 be deterministic, $A_1 \prec A_2$ with the simulation relation R_1, $A_2 \prec A_1$ with the simulation relation R_2 and suppose that $R_1^{-1} = R_2$ (and hence $A_1 \cong A_2$), but there does not exist a deterministic automaton A_1' such that $A_1' \cong A_2$. This means that $\exists s \in E^*$, $\sigma \in E$, $y_1, y_2 \in Y$, $y_1 \in \delta_2(y_0, s)$, $y_2 \in \delta_2(y_0, s)$, $\delta_2(y_1, \sigma)!$, but $\neg\delta_2(y_2, \sigma)!$. From $A_2 \prec A_1$, $y_1 \in \delta_2(y_0, s) \wedge \delta_2(y_1, \sigma)!$ it implies that $\exists x_1 \in X$, $\delta_1(x_0, s) = x_1 \wedge \delta_1(x_1, \sigma)!$. On the other hand, A_1 is deterministic; hence, $\forall x_2 \in X$, $\delta_1(x_0, s) = x_2 \Rightarrow x_2 = x_1$. Therefore, $A_2 \prec A_1$ necessarily leads to $(y_2, x_1) \in R_2$. But, $\exists \sigma \in E$ such that $\delta_1(x_1, \sigma)! \wedge \neg\delta_2(y_2, \sigma)!$, meaning that $(y_2, x_1) \in R_2 \wedge (x_1, y_2) \notin R_1$, i.e., $R_1^{-1} \neq R_2$, that contradicts with the hypothesis, and the necessity is followed. ∎

Next, let A_1 and A_2 to be substituted by A_S and $P_1(A_S)||P_2(A_S)$, respectively, in Lemma 2.7. Then, the existence of $A_1' = A_S'$ in Lemma 2.7 is characterized by the following lemma.

Lemma 2.8 *Consider a deterministic automaton A_S and its natural projections $P_i(A_S)$, $i = 1, 2$. Then, there exists a deterministic automaton A_S' such that $A_S' \cong P_1(A_S)||P_2(A_S)$ if and only if there exist deterministic automata $P_i'(A_S)$ such that $P_i'(A_S) \cong P_i(A_S)$, $i = 1, 2$.*

Proof: Let $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$, $P_i(A_S) = (Q_i, q_0^i, E_i, \delta_i)$, $P_i'(A_S) = (Q_i', q_{0,i}', E_i, \delta_i')$, $i = 1, 2$, $P_1(A_S)||P_2(A_S) = (Z, z_0, E, \delta_{||})$, $P_1'(A_S)||P_2'(A_S) =$

$(Z', z'_0, E, \delta'_{||})$. The proof of Lemma 2.8 is then presented as follows.

Sufficiency: The existence of deterministic automata $P'_i(A_S)$ such that $P'_i(A_S) \cong P_i(A_S)$, $i = 1, 2$ implies that δ'_1 and δ'_2 are functions, and consequently from definition of parallel composition (Definition 1.9), $\delta'_{||}$ is a function, and hence $P'_1(A_S)||P'_2(A_S)$ is deterministic. Moreover, from Lemma 2.6, $P'_i(A_S) \cong P_i(A_S)$, $i = 1, 2$ leads to $P'_1(A_S)||P'_2(A_S) \cong P_1(A_S)||P_2(A_S)$, meaning that there exists a deterministic automaton $A'_S = P'_1(A_S)||P'_2(A_S)$ such that $A'_S \cong P_1(A_S)||P_2(A_S)$.

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 $i = 1$ or $i = 2$, then there does not exist a deterministic automaton A'_S such that $A'_S \cong P_1(A_S)||P_2(A_S)$.

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 $\exists q, q_1, q_2 \in Q$, $e \in E_1$, $t_1, t_2 \in (E_2 \backslash E_1)^*$, $t \in E^*$, $\delta(q, t_1e) = q_1$, $\delta(q, t_2e) = q_2$, $\delta(q_1, t)!$, but $\nexists t' \in E^*$, $\delta(q_2, t')!$, such that $p_1(t) = p_1(t')$, that particularly results in $\delta(q_1, t)! \wedge \neg\delta(q_2, t')!$. From $\delta(q_1, t)! \wedge \neg\delta(q_2, t')!$, the definition of natural projection and Lemma 2.4, it follows that $[q_1]_1 \in \delta_1([q]_1, e)$, $\delta_1([q_1]_1, p_1(t))!$, $[q_2]_1 \in \delta_1([q]_1, e)$, $\neg\delta_1([q_2]_1, p_1(t))!$, $[q_1]_2 \in \delta_2([q]_2, t_1p_2(e))$, $\delta_2([q_1]_2, p_2(t))!$, $([q_1]_1, [q_1]_2) \in \delta_{||}(([q]_1, [q]_2), t_1e)$, $\delta_{||}(([q_1]_1, [q_1]_2), t)!$; whereas, $([q_2]_1, [q_1]_2) \in \delta_{||}(([q]_1, [q]_2), t_1e)$, $\neg\delta_{||}(([q_2]_1, [q_1]_2), t)!$, implying that there does not exist a deterministic automaton A'_S such that $A'_S \cong P_1(A_S)||P_2(A_S)$, and the necessity is proven. ∎

Now, Lemma 2.3 is proven as follows.

Sufficiency: $DC4$ implies that there exist deterministic automata $P_i'(A_S)$ such that $P_i'(A_S) \cong P_i(A_S)$, $i = 1, 2$. Then, from Lemmas 2.6 and 2.8, it follows, respectively, that $P_1'(A_S)||P_2'(A_S) \cong P_1(A_S)||P_2(A_S)$, and that there exists a deterministic automaton $A_S' = P_1'(A_S)||P_2'(A_S)$ such that $A_S' \cong P_1(A_S)||P_2(A_S)$ that due to Lemma 2.7, it results in $R_1^{-1} = R_2$.

Necessity: Let A_S be deterministic, $A_S \prec P_1(A_S)||P_2(A_S)$ with the simulation relation R_1 and $P_1(A_S)||P_2(A_S) \prec A_S$ 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 $i = 1$ or $i = 2$, 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 2.8, there does not exist a deterministic automaton A_S' such that $A_S' \cong P_1(A_S)||P_2(A_S)$; hence, according to Lemma 2.7, it leads to $R_1^{-1} \neq R_2$ which is a contradiction; hence, the necessity is proven.

Chapter 3

Cooperative Tasking for n Agents

3.1 Introduction

This chapter continues the work in Chapter 2 and generalizes the proposed top-down cooperative control into an arbitrary finite number of agents. As the first attempt, this chapter proposes a hierarchical algorithm as a sufficient condition. If the algorithm proceeds up to the end to complete the decomposition, then, the satisfaction of local specifications leads to the satisfaction of the global specification by the team. However, the hierarchical approach depends on the order of local event sets to be chosen in each stage for decomposition. Moreover, since it is a sufficient condition only, there will be no conclusion on the task decomposability, when the hierarchical algorithm does not proceed for decomposition. Therefore, it would be advantageous if the necessary and sufficient decomposability conditions could be developed for an arbitrary finite number of agents. The conditions would be able to check the decomposability, in one stage. Moreover, if the conditions are also necessary, then violation of one of them will result in the indecomposability of the global specification. For this purpose, this chapter provides necessary and sufficient conditions for the decomposability of a global task automaton, with respect to an arbitrary finite number of agents, based on the capability of the agents on decision making on the order and selection of events, the interleaving of synchronized strings and the determinism of bisimulation quotients of local task automata. Moreover, together with the properties of parallel composition, it is also proven that if local controller automata exist such that local task automata are satisfied in the sense of bisimulation, the global task automaton will be guaranteed to be satisfied by the team of agents.

The rest of the chapter is organized as follows. As the first step, Section 3.2, introduces the algorithm for the hierarchical extension of the automaton decomposition for more than two agents. This section also shows that if local controllers exit to enforce local task automata, the entire closed loop system will satisfy the global specification. To illustrate the hierarchical task decomposition, an implementation result is given on a cooperative multi-robot system example in Section 3.3. Afterwards, Section 3.4 discusses that the hierarchical algorithm is sufficient only, and consequently, 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. The cooperative multi-robot system example is also revisited in this section to be handled by the proposed necessary and sufficient conditions. This section also discusses two special cases. Firstly, when an agent has all events of the global event set, it serves as a centralized decision maker for all orders and selections, and the conditions on decision making will be relaxed. Secondly, when the event set can be partitioned into mutual exclusive clusters of local event sets, such that the task automaton is decomposable from the perspective of each cluster, then the decomposability of the global task automaton is reduced to the global decision making on the orders and selections between transitions. Finally, the chapter concludes with remarks and discussions in

Section 3.5. The proofs of lemmas and propositions are given in the Appendix.

3.2 Hierarchical Decomposition

The previous chapter addressed Problem 2.1 and presented necessary and sufficient conditions for the decomposability of an automaton with respect to the parallel composition and two local event sets. However, in practice, multi-agent systems are typically comprised of many local agents that work as a team. To generalize the result, a hierarchical algorithm is proposed here to the case of arbitrary finite number of agents. Consider a task automaton A_S to be decomposed with respect to parallel composition and local event sets E_i, $i = 1, 2, ..., n$, so that $E = \overset{n}{\underset{i=1}{\cup}} E_i$. We propose the following algorithm as a sufficient condition for the hierarchical decomposition of the given task automaton.

Algorithm 3.1 *(Hierarchical decomposition algorithm)*

1. $E = \overset{n}{\underset{i=1}{\cup}} E_i$, $\Sigma = \{E_1, ..., E_n\}$, $K = \{1, ..., n\}$.
2. $i = 1$, *find* $k \in K$ *such that* $\Sigma_i = E_k \in \Sigma$, $\bar{\Sigma}_i = \underset{j \in K \backslash k}{\cup} E_j$, *so that* A_S *satisfies decomposability conditions* $DC1$-$DC4$ *in Theorem* 2.1, *i.e.,* $A_S \cong P_{\Sigma_i}(A_S) || P_{\bar{\Sigma}_i}(A_S)$.
3. $K = K \backslash k$, $\Sigma = \{E_j\}_{j \in K}$, $A_S = P_{\bar{\Sigma}_i}(A_S)$, $i = i + 1$, *go to Step* 2.
4. *Continue until* $i = n - 1$ *or no more decomposition is possible in* $i = m - 1$, $m \leq n$. *Then* $\Sigma_m = \bar{\Sigma}_{m-1}$; *hence,* A_S *is decomposable with respect to parallel composition and natural projections into* $\{\Sigma_1, \cdots, \Sigma_m\} \subseteq \Sigma$, *if the algorithm*

proceeds up to $i = m - 1$.

Remark 3.1 The computational complexity of the algorithm in the worst case is of order $O(n^2(|E|^2 \times |Q| \times \kappa + \sum\limits_{a\in E_1\cap E_2} |p_a(L(A_S))|^2))$, where $\kappa = \underset{t\in L(A_S)}{max} |t|$, assuming the number of appearing events as the length of loops. In practice, during the iterations, $|E|$ is replaced by $\overset{|K|}{\underset{j=1}{\cup}} E_j$ which is decreasing with respect to iteration. Moreover, the second term in the complexity expression shows that the less number of common events and the less appearance of common events in A_S, the less complexity. The algorithm will terminate due to finite number of states and local event sets. If the algorithm successfully proceeds to step $n-1$, the automaton A_S is decomposable and we obtain a complete decomposition of the global specification into subtasks for each individual agent. However, it is unclear whether the algorithm can successfully terminate for any decomposable task automaton (necessity).

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

Problem 3.1 *(Cooperative Tasking Problem) Consider a plant, represented by a parallel distributed system $\overset{n}{\underset{i=1}{\|}} A_{P_i}$, with local event sets E_i, $i = 1, ..., n$, and let the global specification is given by a decomposable deterministic task automaton A_S, where $A_S \cong \overset{n}{\underset{i=1}{\|}} P_i(A_S)$ with $E = \overset{n}{\underset{i=1}{\cup}} E_i$. Then, does designing local controllers A_{C_i}, so that $A_{C_i} \| A_{P_i} \cong P_i(A_S)$, $i = 1, \cdots, n$, derive the global closed loop system to satisfy the global specification A_S, in the sense of bisimilarity, i.e., $\overset{n}{\underset{i=1}{\|}} (A_{C_i} \| A_{P_i}) \cong A_S$?*

Following result provides a sufficient condition for Problem 3.1, using the hierarchical decomposition method.

Theorem 3.1 *(Hierarchical Cooperative Tasking) Consider a plant, represented by a deterministic parallel distributed system* $A_P = \overset{n}{\underset{i=1}{\parallel}} A_{P_i}$*, with given local event sets* E_i*,* $i = 1, ..., n$*, and given specification represented by a deterministic automaton* A_S*, with* $E = \overset{n}{\underset{i=1}{\cup}} E_i$*. If the Algorithm 3.1 continues up to* $i = n - 1$*, and there exist local controllers* A_{C_i}*, so that* $A_{C_i} \parallel A_{P_i} \cong P_i(A_S)$*,* $i = 1, 2, \cdots, n$*, then the global closed loop system satisfies the global specification* A_S*, in the sense of bisimilarity, i.e.,* $\overset{n}{\underset{i=1}{\parallel}} (A_{C_i} \parallel A_{P_i}) \cong A_S$*.*

Proof: Algorithm 3.1 is a direct extension of Theorem 2.1 combined with the associativity property of parallel decomposition [46]. Then, provided $DC1$-$DC4$ in each step and choosing local controllers A_{C_i}, so that $A_{C_i} \parallel A_{P_i} \cong P_i(A_S)$, $i = 1, 2, \cdots, n$, due to Lemma 2.6 leads to $\overset{n}{\underset{i=1}{\parallel}} (A_{C_i} \parallel A_{P_i}) \cong \overset{n}{\underset{i=1}{\parallel}} P_i(A_S) \cong A_S$. ■

Now, if $DC1$-$DC4$ are reduced to $DC1$-$DC3$ (Theorem 2.1 is reduced into Lemma 2.2), then $\overset{n}{\underset{i=1}{\parallel}} P_i(A_S) \cong A_S$ is reduced into $\overset{n}{\underset{i=1}{\parallel}} P_i(A_S) \prec A_S$; hence, choosing local controllers A_{C_i}, so that $A_{C_i} \parallel A_{P_i} \prec P_i(A_S)$, $i = 1, 2, \cdots, n$, due to Lemma 2.6 leads to $\overset{n}{\underset{i=1}{\parallel}} (A_{C_i} \parallel A_{P_i}) \prec \overset{n}{\underset{i=1}{\parallel}} P_i(A_S) \prec A_S$. Therefore,

Corollary 3.1 *Considering the plant and global task as stated in Theorem 3.1, if* $DC1$*-*$DC4$ *is reduced to* $DC1$*-*$DC3$ *in Algorithm 3.1 and it continues up to* $i = n-1$*, then the existence of local controllers* A_{C_i}*, so that* $A_{C_i} \parallel A_{P_i} \prec P_i(A_S)$*,* $i = 1, 2, \cdots, n$*, derive the global closed loop system to satisfy the global specification* A_S*, in the sense of similarity, i.e.,* $\overset{n}{\underset{i=1}{\parallel}} (A_{C_i} \parallel A_{P_i}) \prec A_S$*.*

3.3 Example

Consider the cooperative multi-robot system in Examples 1.3 and Figure 1.4.

We check decomposability condition for this global task automaton with respect to $\Sigma_1 = E_2$ and $\bar{\Sigma}_1 = E_1 \cup E_3$ for the first stage in Algorithm 1. Firstly, $\{h_2, \ R_2to2, R_2in2\} \subset \Sigma_1 \backslash \bar{\Sigma}_1$ can occur in any order with respect to $\{h_1, R_1toD_1, R_1onD_1, h_3, R_3to3, R_3in3, R_3toD_1, R_3onD_1, FWD\} = \bar{\Sigma}_1 \backslash \Sigma_1$, as it is shown in the global automaton in Figure 1.5, satisfying $DC1$ and $DC2$. Moreover, $D_1opened$, R_2in1 and r are common events, provided by R_1, R_2 and R_3, respectively, and informed to other two robots upon occurrence. Since $\{D_1opened, FWD\} \subseteq \bar{\Sigma}_1$, $\{R_2in2, D_1opened\} \subseteq \Sigma_1$, $\{D_1opened, \ R_2to1\} \subseteq \Sigma_1$, $\{R_2to1, R_2in1\} \subseteq \Sigma_1$, $\{R_2in1, BWD\} \subseteq \bar{\Sigma}_1$, $\{BWD, D_1closed\} \subseteq \bar{\Sigma}_1$, $\{D_1closed, R_3to1\} \subseteq \bar{\Sigma}_1$, $\{R_3to1, R_3in1\} \subseteq \bar{\Sigma}_1$, $\{R_3in1, r\} \subseteq \bar{\Sigma}_1$, $\{r, h_1\} \subseteq \bar{\Sigma}_1$, $\{r, h_2\} \subseteq \Sigma_1$, $\{r, h_3\} \subseteq \bar{\Sigma}_1$; hence, all these successive transitions satisfy $DC1$ and $DC2$. Furthermore, all common events $\{D_1opened, R_2in1, r\} \subseteq \Sigma_1 \cap \bar{\Sigma}_1$ appear in only one branch; hence, $DC3$ is satisfied. Finally, $P_{\Sigma_1}(A_S)$ and $P_{\bar{\Sigma}_1}(A_S)$ are both deterministic; hence, $DC4$ is satisfied. Therefore, due to Theorem 2.1, A_S can be decomposed into $P_2(A_S) = P_{\Sigma_1}(A_S) = A_2$ and $P_{1,3}(A_S) := P_{E_1 \cup E_3}(A_S) = P_{\bar{\Sigma}_1}(A_S)$.

For the second stage, the private transitions defined over $E_1 \backslash E_3 = \{h_1, R_1toD_1, R_1onD_1\}$ can occur in any order with respect to the transitions defined over the private local event set $\{h_3, R_3to3, R_3in3, R_3toD_1, R_3onD_1\} \subset E_3 \backslash E_1$. Moreover, $\{R_3onD_1, FWD\} \subseteq E_3$, $\{R_1onD_1, FWD\} \subseteq E_1$, $\{FWD, D_1opened\} \subseteq E_1 \cap E_3$, $\{D_1opened, R_2in1\} \subseteq E_1 \cap E_3$, $\{R_2in1, BWD\} \subseteq E_1 \cap E_3$, $\{BWD, \ D_1closed\} \subseteq E_1 \cap$

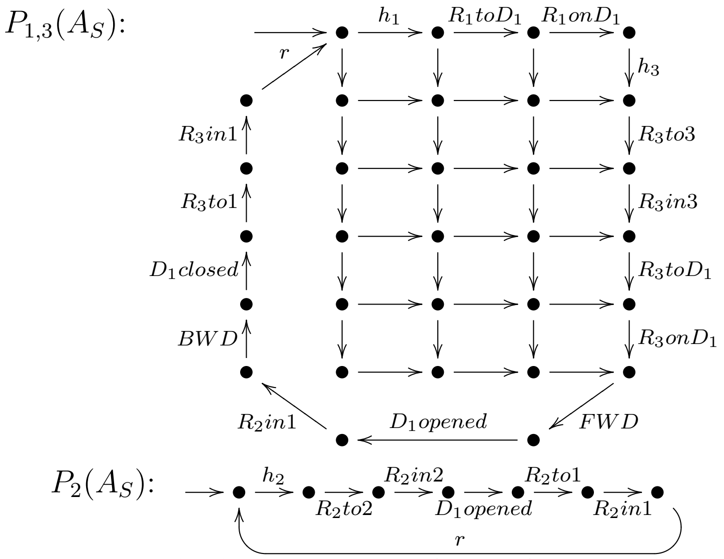


Figure 3.1: $P_{E_1 \cup E_3}(A_S)$ for the team $\{R_1, R_3\}$ and $P_2(A_S)$ for R_2.

E_3, $\{D_1closed, R_3to1\} \subseteq E_3$, $\{R_3to1, R_3in1\} \subseteq E_3$, $\{R_3in1, r\} \subseteq E_3$, $\{r, h_3\} \subseteq E_3$ and $\{r, h_1\} \subseteq E_1$. Therefore, $DC1$ and $DC2$ are satisfied. Furthermore, since all common events $\{FWD, D_1opened, R_2in1, BWD,\ D_1closed, r\} \subseteq E_1 \cap E_3$ appear in only one branch in $P_{1,3}(A_S)$, therefore, there are no pairs of strings violating $DC3$; therefore, $DC3$ is also satisfied. Moreover, $P_1(A_S)$ and $P_3(A_S)$ are both deterministic, and consequently, $P_{1,3}(A_S)$ satisfies $DC4$. Therefore, $P_{1,3}(A_S)$ is decomposable into $P_1(A_S)$ and $P_3(A_S)$. The results of two decomposition stages are shown in Figures 3.1 and 1.6, such that $P_1(A_S) \parallel P_2(A_S) \parallel P_3(A_S) \cong A_S$.

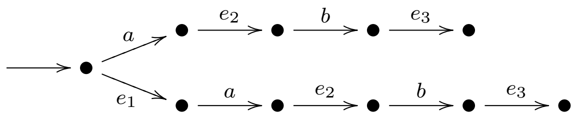
This scenario has been successfully implemented on a team of three ground robots, shown in Figure 1.4.

3.4 Necessary and Sufficient Decomposability Conditions for n Agents

3.4.1 Link to the result for two agents

This part proposes necessary and sufficient conditions for task automaton decomposability problem (Problem 2.1), for an arbitrary finite number of agents.

Theorem 2.1 showed the decomposability of an automaton with respect to the parallel composition and two local event sets. However, in practice, multi-agent systems are typically comprised of many individual agents that work as a team. For this purpose, previous section introduced a hierarchical decomposition method to handle only two individual event sets at each step for partitioning: an event set; and the set of the rest of event sets. If the algorithm successfully terminates, then the decompositions can be obtained, and subsequently, the proposed top-down approach can by applied, however, the algorithm depends strongly on the order of the event sets that are chosen for decomposition (see Example 3.1 as follows). In addition, as it is illustrated in Example 3.2, the algorithm is sufficient only, which means that if the algorithm successfully terminates, then A_S is decomposable, otherwise, there is no conclusion on its decomposability.

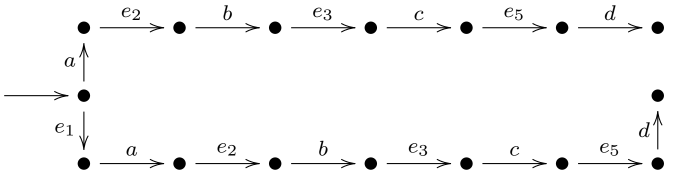
Example 3.1 Consider A_S: 

with $E = E_1 \cup E_2 \cup E_3$, $E_1 = \{a, e_1\}$, $E_2 = \{a, b, e_2\}$, $E_3 = \{b, e_3\}$. This automaton is decomposable as $A_S \cong P_1(A_S)||P_2(A_S)||P_3(A_S) \cong P_1(A_S)||(P_2(A_S)||P_3(A_S)) \cong$

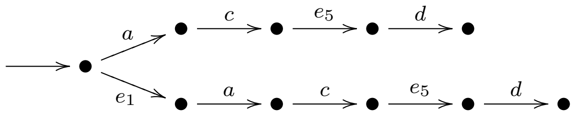
$P_3(A_S)||(P_1(A_S)||P_2(A_S)) \cong P_2(A_S)||(P_1(A_S)||P_3(A_S))$. In this example, $P_{E_2\cup E_3}(A_S) \cong P_2(A_S)||P_3(A_S)$ and $P_{E_1\cup E_2}(A_S) \cong P_1(A_S)||P_2(A_S)$, but $P_{E_1\cup E_3}(A_S) \ncong P_1(A_S)||P_3(A_S)$. This means that while choosing $P_1(A_S)$ or $P_3(A_S)$ as the first set allows the hierarchical algorithm to proceeds up to $A_S \cong P_1(A_S)||P_2(A_S)||P_3(A_S)$, choosing $P_2(A_S)$ will stuck the algorithm in the second step as $A_S \cong P_2(A_S)||P_{E_1\cup E_3}(A_S)$, but $P_{E_1\cup E_3}(A_S) \ncong P_1(A_S)||P_3(A_S)$.

Example 3.2

The automaton A_S:



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



, $P_2(A_S) \cong \to\bullet\xrightarrow{a}\bullet\xrightarrow{e_2}\bullet\xrightarrow{b}\bullet\xrightarrow{d}\bullet$ and $P_3(A_S) \cong \to\bullet\xrightarrow{b}\bullet\xrightarrow{e_3}\bullet\xrightarrow{c}\bullet$, is decomposable as $A_S \cong P_1(A_S)||P_2(A_S)||P_3(A_S) \cong P_1(A_S)||(P_2(A_S)||P_3(A_S)) \cong P_3(A_S)||(P_1(A_S)||P_2(A_S)) \cong P_2(A_S)||(P_1(A_S)||P_3(A_S))$, although $P_{E_2\cup E_3}(A_S) \ncong P_2(A_S)||P_3(A_S)$, $P_{E_1\cup E_2}(A_S) \ncong P_1(A_S)||P_2(A_S)$ and $P_{E_1\cup E_3}(A_S) \ncong P_1(A_S)||P_3(A_S)$. This means that although A_S is decomposable with respect to $P_1(A_S)$, $P_2(A_S)$ and $P_3(A_S)$, choosing any of local event sets E_1, E_2 and E_3 passes the first stage of the hierarchical decomposition, but the algorithm will stuck at the second step.

Therefore, it would be very advantageous if we can find necessary and sufficient conditions for the decomposability of a deterministic automaton with respect to an arbitrary finite number of local event sets. The proposed conditions should be independent of the order of the local event sets and should give us more insights on the cooperative tasking among agents. The main efforts in the rest of this chapter are to generalize the decomposability conditions $DC1$-$DC4$ for an arbitrary finite number of agents.

The first difficulty is to generalize $DC1$ and $DC2$ for more than two agents. For two-agent case, $DC1$ and $DC2$ rely on the notion of "private events from different local event sets" ($e_1 \in E_1 \backslash E_2$ and $e_2 \in E_2 \backslash E_1$), namely, an event is either private (belongs to the union but not the intersection of local event sets) or common (belongs to both local event sets). For more than two agents, on the other hand, the notion of common event should be replaced by the notion of "shared events", namely, the events that belong to more than one agent. Accordingly, the first two decomposability conditions are required to be restated based on the notion of shared events, as the following lemma. In this perspective, the first two conditions say that for any decision on selection (defined by adjacent events) or decision on the order (defined by successive events) there should exist at least one agent capable of the decision making (at least one local event set containing both events) or the decision is not important (both orders are legal).

The third condition, $DC3$, also is not easily generalizable, and a more restrictive condition is given for n agents. For two-agent case, the pairs of identical strings sharing common events were excluded as they were already checked by the first two conditions. For more than two agents, however, due to difficulty in the exclusion of different permutation of strings, a more restrictive condition is introduced, in the sense that the third condition may check some strings that have been already checked by the first two conditions on the selection and order decision makings. Nevertheless, the new conditions are necessary and sufficient conditions for an arbitrary finite number of agents and do not depend on the order of agents. Moreover, to mitigate the restrictiveness of the third condition, it will be shown that for mutually exclusive clusters of event sets the third and fourth conditions are required to be checked only for each cluster.

Once $DC1$-$DC3$ are generalized, the fourth condition $DC4$ is already in the form that can be generalized for any number of agents, stating that for any local task automaton, there should exist a deterministic bisimilar automaton.

Before stating the main result, firstly, an alternative representation on $DC1$ and $DC2$ is given in the following lemma, by replacing $\forall e_1 \in E_1 \backslash E_2, e_2 \in E_2 \backslash E_1$ (two events from different local event sets) with $\forall e_1, e_2 \in E, \forall E_i \in \{E_1, E_2\}, \{e_1, e_2\} \not\subset E_i$ (two independent events that no local event set contains both of them). This provides a way to generalize the first two decomposability conditions for more than two agents.

Lemma 3.1 *Consider a deterministic automaton $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$ and natural projections P_i, $i = 1, 2$. Then the following two cases are equivalent*

1. $\forall e_1 \in E_1 \backslash E_2, e_2 \in E_2 \backslash E_1, q \in Q$, $s \in E^*$,

   - *$DC1$:* $[\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*
   - *$DC2$:* $\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!$

2. $\forall e_1, e_2 \in E, \forall E_i \in \{E_1, E_2\}, \{e_1, e_2\} \not\subset E_i, q \in Q$, $s \in E^*$*:*

- *DC*1*:* $[\delta(q,e_1)! \wedge \delta(q,e_2)!] \Rightarrow [\delta(q,e_1e_2)! \wedge \delta(q,e_2e_1)!]$*;*
- *DC*2*:* $\delta(q,e_1e_2s)! \Leftrightarrow \delta(q,e_2e_1s)!$

Proof: See the proof in the Appendix. ■

The decomposability conditions for an arbitrary finite number of agents will be then given in the next part.

3.4.2 Necessary and Sufficient Decomposability Conditions for n Agents

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

Firstly, $\overset{n}{\underset{i=1}{||}} P_i(A_S)$ always simulates A_S. Formally:

Lemma 3.2 *Consider a deterministic automaton* $A_S = \left(Q, q_0, E = \bigcup\limits_{i=1}^{n} E_i, \delta\right)$ *and natural projections* P_i*,* $i = 1, ..., n$*. Then, it always holds that* $A_S \prec \overset{n}{\underset{i=1}{||}} P_i(A_S)$*.*

Proof: See the proof in the Appendix. ■

The similarity of $\overset{n}{\underset{i=1}{||}} P_i(A_S)$ to A_S, however, is not always true (see Examples 3.5, 3.6, 3.8 and 3.9), and needs some conditions as stated in the following lemma.

Lemma 3.3 *Consider a deterministic automaton* $A_S = \left(Q, q_0, E = \bigcup\limits_{i=1}^{n} E_i, \delta\right)$ *and natural projections* P_i*,* $i = 1, ..., n$*. Then,* $\overset{n}{\underset{i=1}{||}} P_i(A_S) \prec A_S$ *if and only if* $\forall e_1, e_2 \in E, q \in Q, s \in E^*, \forall E_i \in \{E_1, ..., E_n\}, \{e_1, e_2\} \not\subset E_i$*:*

- *DC*1*:* $[\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*

- *DC*2*:* $\delta\left(q, e_1e_2s\right)! \Leftrightarrow \delta\left(q, e_2e_1s\right)!$*;*

- *DC*3*:* $\delta(q_0, \overset{n}{\underset{i=1}{|}} \overline{p_i(s_i)})!$*,* $\forall\{s_1, \cdots, s_n\} \in \tilde{L}\left(A_S\right)$*,* $\exists s_i, s_j \in \{s_1, \cdots, s_n\}, s_i \neq s_j$*, where,* $\tilde{L}\left(A_S\right) \subseteq L\left(A_S\right)$ *is the largest subset of* $L\left(A_S\right)$ *such that* $\forall s \in \tilde{L}\left(A_S\right), \exists s' \in \tilde{L}\left(A_S\right),\ \exists E_i, E_j \in \{E_1, ..., E_n\}, i \neq j, p_{E_i\cap E_j}\left(s\right)$ *and* $p_{E_i\cap E_j}\left(s'\right)$ *start with the same event.*

Proof: See the proof in the Appendix. ∎

Next, we need to show that for two simulation relations R_1 (for $A_S \prec \overset{n}{\underset{i=1}{||}} P_i\left(A_S\right)$) and R_2 (for $\overset{n}{\underset{i=1}{||}} P_i\left(A_S\right) \prec A_S$) defined by the previous two lemmas, $R_1^{-1} = R_2$.

Lemma 3.4 *Consider an automaton* $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$ *with natural projections* P_i*,* $i = 1, ..., n$*. If* A_S *is deterministic,* $A_S \prec \overset{n}{\underset{i=1}{||}} P_i\left(A_S\right)$ *with the simulation relation* R_1 *and* $\overset{n}{\underset{i=1}{||}} P_i\left(A_S\right) \prec A_S$ *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 DC*4*:* $\forall i \in \{1, ..., n\}$*,* $x, x_1, x_2 \in Q_i$*,* $x_1 \neq x_2$*,* $e \in E_i$*,* $t \in E_i^*$*,* $x_1 \in \delta_i(x, e)$*,* $x_2 \in \delta_i(x, e)$*:* $\delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$.

Proof: See the proof in the Appendix. ∎

Based on these lemmas, the main result on task automaton decomposability is given as follows.

Theorem 3.2 *(Task Decomposability for n Agents) A deterministic automaton* $A_S = \left(Q, q_0, E = \bigcup\limits_{i=1}^{n} E_i, \delta\right)$ *is decomposable with respect to parallel composition and natural projections* P_i*,* $i = 1, ..., n$ *such that* $A_S \cong \overset{n}{\underset{i=1}{||}} P_i\left(A_S\right)$ *if and only if* A_S *satisfies* *the following decomposability conditions (DC):* $\forall e_1, e_2 \in E, q \in Q, s \in E^*, \forall E_i \in \{E_1, ..., E_n\}, \{e_1, e_2\} \not\subset E_i$*:*

- *DC*1*:* $[\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*
- *DC*2*:* $\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!$*;*
- *DC*3*:* $\delta(q_0, \mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)})!$*,* $\forall\{s_1, \cdots, s_n\} \in \tilde{L}(A_S)$*,* $\exists s_i, s_j \in \{s_1, \cdots, s_n\}, s_i \neq s_j$*, where,* $\tilde{L}(A_S) \subseteq L(A_S)$ *is the largest subset of* $L(A_S)$ *such that* $\forall s \in \tilde{L}(A_S), \exists s' \in \tilde{L}(A_S),\ \exists E_i, E_j \in \{E_1, ..., E_n\}, i \neq j, p_{E_i \cap E_j}(s)$ *and* $p_{E_i \cap E_j}(s')$ *start with the same event, and*
- *DC*4*:* $\forall i \in \{1, ..., n\}$*,* $x, x_1, x_2 \in Q_i$*,* $x_1 \neq x_2$*,* $e \in E_i$*,* $t \in E_i^*$*,* $x_1 \in \delta_i(x, e)$*,* $x_2 \in \delta_i(x, e)$*:* $\delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$*.*

Proof: According to Definition 1.11, $A_S \cong \mathop{||}\limits_{i=1}^{n} P_i(A_S)$ if and only if $A_S \prec \mathop{||}\limits_{i=1}^{n} P_i(A_S)$ (that is always true due to Lemma 3.2), $\mathop{||}\limits_{i=1}^{n} P_i(A_S) \prec A_S$ (that it is true if and only if $DC1$, $DC2$ and $DC3$ are true, according to Lemma 3.3) and $R_1^{-1} = R_2$ (that for a deterministic automaton A_S, when $A_S \prec \mathop{||}\limits_{i=1}^{n} P_i(A_S)$ with simulation relation R_1 and $\mathop{||}\limits_{i=1}^{n} P_i(A_S) \prec A_S$ with simulation relation R_2, due to Lemma 3.4, $R_1^{-1} = R_2$ holds true if and only if $DC4$ is satisfied). Therefore, $A_S \cong \mathop{||}\limits_{i=1}^{n} P_i(A_S)$ if and only if $DC1$, $DC2$, $DC3$ and $DC4$ are satisfied. ■

A more insightful set of expressions for $DC1$ and $DC2$ are presented in the following lemma.

Lemma 3.5 *Consider a deterministic automaton* $A_S = (Q, q_0, E = \bigcup\limits_{i=1}^{n} E_i, \delta)$ *and natural projections* P_i*,* $i = 1, ..., n$*. Then following statements are equivalent*

1. $\forall e_1, e_2 \in E, q \in Q, s \in E^*, \forall E_i \in \{E_1, ..., E_n\}, \{e_1, e_2\} \not\subset E_i$*:*

- *DC*1*:* $[\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*
- *DC*2*:* $\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!$*;*

2.
- *DC*1*:* $\forall e_1, e_2 \in E, q \in Q$*:* $[\delta(q, e_1)! \wedge \delta(q, e_2)!]$

  $\Rightarrow [\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*
- *DC*2*:* $\forall e_1, e_2 \in E, q \in Q,\ s \in E^*$*:* $[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!]$

  $\Rightarrow [\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$.

Proof: See the proof in the Appendix. ∎

The alternative statements for $DC1$ and $DC2$ in Lemma 3.5, clearly state that any decision on any selection or order between two transitions should be either possible by at least one of the agents, or otherwise, the decision should not be important, in the sense that both permutations be legal. This lemma together with Theorem 3.2 give an alternative result for decomposability conditions as follows.

Corollary 3.2 *A deterministic automaton* $A_S = \left(Q, q_0, E = \bigcup_{i=1}^{n} E_i, \delta\right)$ *is decomposable with respect to parallel composition and natural projections* P_i*,* $i = 1, ..., n$ *such that* $A_S \cong \mathop{\|}_{i=1}^{n} P_i(A_S)$ *if and only if* A_S *satisfies the following decomposability conditions (DC):*

- *DC*1*:* $\forall e_1, e_2 \in E, q \in Q$*:* $[\delta(q, e_1)! \wedge \delta(q, e_2)!]$

  $\Rightarrow [\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*
- *DC*2*:* $\forall e_1, e_2 \in E, q \in Q,\ s \in E^*$*:* $[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!]$

  $\Rightarrow [\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$*;*

- *DC*3: $\delta(q_0, \overset{n}{\underset{i=1}{|}} \overline{p_i(s_i)})!$, $\forall \{s_1, \cdots, s_n\} \in \tilde{L}(A_S)$, $\exists s_i, s_j \in \{s_1, \cdots, s_n\}, s_i \neq s_j$, *where,* $\tilde{L}(A_S) \subseteq L(A_S)$ *is the largest subset of* $L(A_S)$ *such that* $\forall s \in \tilde{L}(A_S), \exists s' \in \tilde{L}(A_S),\ \exists E_i, E_j \in \{E_1, ..., E_n\}, i \neq j, p_{E_i \cap E_j}(s)$ *and* $p_{E_i \cap E_j}(s')$ *start with the same event, and*

- *DC*4: $\forall i \in \{1, ..., n\}$, $x, x_1, x_2 \in Q_i$, $x_1 \neq x_2$, $e \in E_i$, $t \in E_i^*$, $x_1 \in \delta_i(x, e)$, $x_2 \in \delta_i(x, e)$: $\delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$.

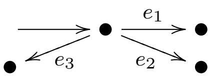
Remark 3.2 Intuitively, the decomposability condition $DC1$ means that for any decision on the selection between two transitions there should exist at least one agent that is capable of the decision making, or the decision should not be important (both permutations in any order be legal). $DC2$ says 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 the 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, $DC3$ ensures that an illegal behavior (a string that does not appear in A_S) is not allowed by the team (does not appear in $\overset{n}{\underset{i=1}{||}} P_i(A_S)$). 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 $\overset{n}{\underset{i=1}{||}} P_i(A_S)$ 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 $\overset{n}{\underset{i=1}{||}} P_i(A_S)$).

The following examples illustrate the decomposability conditions for decomposable and indecomposable automata.

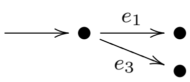
Example 3.3 Consider the automaton A_S in Example 3.1. We denote the string on the top and bottom branches of A_S as s_1 and s_2, respectively. Since $\{e_1, a\} \subseteq E_1$, $\{a, e_2\} \subseteq E_2$, $\{e_2, b\} \subseteq E_2$, $\{b, e_3\} \subseteq E_3$, then $DC1$ and $DC2$ are satisfied. Moreover, s_1, s_2 contain $a \in E_1 \cap E_2$, $b \in E_2 \cap E_3$ where a appears as the first event in both $p_{E_1 \cap E_2}(s_1)$ and $p_{E_1 \cap E_2}(s_2)$. Then, according to $DC3$, following six interleaving transitions are checked: $\delta(q_0, \overline{p_1(s_1)}|\overline{p_2(s_1)}|\overline{p_3(s_2)})!$; $\delta(q_0, \overline{p_1(s_1)}|\overline{p_2(s_2)}|\overline{p_3(s_1)})!$; $\delta(q_0, \overline{p_1(s_1)}|\overline{p_2(s_2)}|\overline{p_3(s_2)})!$; $\delta(q_0, \overline{p_1(s_2)}|\overline{p_2(s_1)}|\overline{p_3(s_1)})!$; $\delta(q_0, \overline{p_1(s_2)}|\overline{p_2(s_1)}|\overline{p_3(s_2)})!$, and $\delta(q_0, \overline{p_1(s_2)}|\overline{p_2(s_2)}|\overline{p_3(s_1)})!$, i.e., $DC3$ is satisfied (note that the expression $\forall \{s_1, \cdots, s_n\} \in \tilde{L}(A_S)$, $\exists s_i, s_j \in \{s_1, \cdots, s_n\}, s_i \neq s_j$ in the statements of $DC3$ excludes $\delta(q_0, \overline{p_1(s_1)}|\overline{p_2(s_1)}|\overline{p_3(s_1)})!$ and $\delta(q_0, \overline{p_1(s_2)}|\overline{p_2(s_2)}|\overline{p_3(s_2)})!$ from the checklist of $DC3$). In addition, since $P_i(A_S)$, $i = 1, 2, 3$, have deterministic bisimilar automata, then $DC4$ is also fulfilled; therefore, A_S is decomposable.

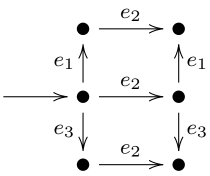
Example 3.4 (Revisiting Example in Section 3.3 for decomposability using Theorem 3.2) As another example for decomposable automaton, we recall the task automaton of cooperative multi-robot example in Section 3.3, where the global task automaton has been decomposed into local task automata using the hierarchical approach, in two stages. Here, we decompose A_S directly using Theorem 3.2. Firstly, $DC1$ and $DC2$ are satisfied since the pairs of adjacent/successive events, each from one of the sets $\{h_1, R_1toD_1, R_1onD_1\} \subseteq E_1 \backslash \{E_2 \cup E_3\}$, $\{h_2, R_2to2, R_2in2\} \subseteq E_2 \backslash \{E_1 \cup E_3\}$ and $\{h_3, R_3to3, R_3in3, R_3toD_1, R_3onD_1\} \subseteq E_3 \backslash \{E_1 \cup E_2\}$ and also the pairs of event FW, paired with events from $\{h_2, R_2to2, R_2in2\}$ appear in both orders in the automaton. The rest of adjacent/successive 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 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 3.2, A_S is decomposable into $P_i(A_S)$, $i = 1, 2, 3$, as illustrated in Figure 1.6.

Examples 3.3 and 3.4 showed decomposable automata. Examples 3.5, 3.6, 3.8 and 3.9 will illustrate the automata that are indecomposable due to the violation of one of the decomposability conditions $DC1$-$DC4$, respectively, although they satisfy other three conditions.

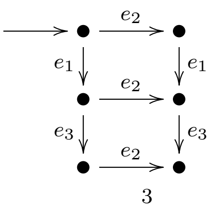
Example 3.5 This example illustrates the concept of decision making on switching between the events, mentioned in Remark 3.2. Furthermore, it shows an automaton that satisfies $DC2$, $DC3$ and $DC4$, but not $DC1$, leading to indecomposability. The automaton A_S: 

with local event sets $E_1 = \{e_1, e_3\}$, $E_2 = \{e_2\}$, $E_3 = \{e_3\}$, is not decomposable as the parallel composition of

$P_1(A_S)$: 

, $P_2(A_S)$: $\rightarrow \bullet \xrightarrow{e_2} \bullet$ and $P_3(A_S)$: $\rightarrow \bullet \xrightarrow{e_3} \bullet$ is obtained as $\overset{3}{\underset{i=1}{||}} P_i(A_S)$: 

which does not bisimulate A_S. Here, $A_S \prec \overset{3}{\underset{i=1}{||}} P_i(A_S)$ but $\overset{3}{\underset{i=1}{||}} P_i(A_S) \not\prec A_S$; hence, $A_S \not\cong \overset{3}{\underset{i=1}{||}} P_i(A_S)$. A_S is not decomposable with respect to parallel composition and natural projections P_i, $i = 1, 2, 3$, since two events e_2 and e_3 and also the pair of e_1 and e_2 do not respect $DC1$, as none of the local plants take in charge of decision making on the switching between these event pairs, as $\delta(q_0, e_2)! \wedge \delta(q_0, e_3)!$, but neither $\exists E_i \in \{E_1, E_2, E_3\}, \{e_2, e_3\} \subseteq E_i$, nor $\delta(q_0, e_2e_3)! \wedge \delta(q_0, e_3e_2)!$. Similarly, $\delta(q_0, e_1)! \wedge \delta(q_0, e_2)!$, but neither $\exists E_i \in \{E_1, E_2, E_3\}, \{e_1, e_2\} \subseteq E_i$, nor $\delta(q_0, e_1e_2)! \wedge \delta(q_0, e_2e_1)$.

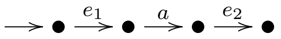
Example 3.6 This example shows the concept of decision making on the order of events, and illustrates an automaton that satisfies $DC1$, $DC3$ and $DC4$, but it is indecomposable because of the violation of $DC2$. Consider the automaton A_S: $\rightarrow \bullet \xrightarrow{e_1} \bullet \xrightarrow{e_2} \bullet \xrightarrow{e_3} \bullet$, with local event sets $E_1 = \{e_1, e_3\}$, $E_2 = \{e_2\}$, $E_3 = \{e_3\}$. The parallel composition of $P_1(A_S)$: $\rightarrow \bullet \xrightarrow{e_1} \bullet \xrightarrow{e_3} \bullet$, $P_2(A_S)$: $\rightarrow \bullet \xrightarrow{e_2} \bullet$ and $P_3(A_S)$: $\rightarrow \bullet \xrightarrow{e_3} \bullet$ is $\overset{3}{\underset{i=1}{||}} P_i(A_S)$: 

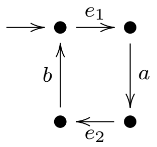
. One can observe that $A_S \prec \overset{3}{\underset{i=1}{||}} P_i(A_S)$ but $\overset{3}{\underset{i=1}{||}} P_i(A_S) \not\prec A_S$; hence, $A_S \not\cong \overset{3}{\underset{i=1}{||}} P_i(A_S)$. Here, A_S respects $DC1$, $DC3$ and $DC4$, but it is indecomposable due to the violation of $DC2$, as none of the local plants take in charge of decision making on the orders of $\{e_1, e_2\}$ and $\{e_2, e_3\}$.

Remark 3.3 Example 3.6 also shows the decomposability of a path automaton (an automaton with no adjacent events) that can be formally stated as a special case of Theorem 3.2 as

Corollary 3.3 *A path automaton (and hence, an automaton with only one self-loop as a special case of a path automaton) is decomposable if and only if any two successive events in the automaton belong to at least one of the local event sets ($\forall q \in Q$, $e_1, e_2 \in E$, $\delta(q, e_1e_2)!$, then $\exists E_i \in \{E_1, \ldots, E_n\}$ such that $\{e_1, e_2\} \subseteq E_i$).*

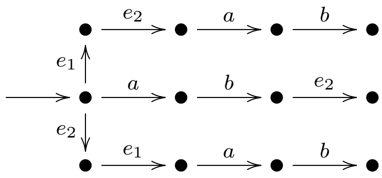
The reason for this corollary is that when the automaton has no adjacent events, the antecedent of $DC1$, $DC3$ and $DC4$ (see Theorem 3.2 and Corollary 3.2) become false and these conditions will be trivially satisfied. Moreover, since there is no adjacent pair of events, the consequence of $DC2$ is reduced to $\exists E_i \in \{E_1, \ldots, E_n\}$, $\{e_1, e_2\} \subseteq E_i$.

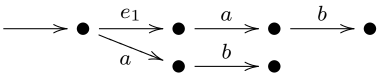
Example 3.7 Two path automata A_1: 

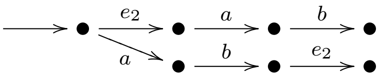
, with local event sets $E_1 = \{e_1, a\}$ and $E_2 = \{e_2, a\}$, and A_2: 

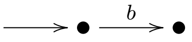
with local event sets $E_1 = \{e_1, a, b\}$ and $E_2 = \{e_2, a, b\}$, are both decomposable according to Corollary 3.3.

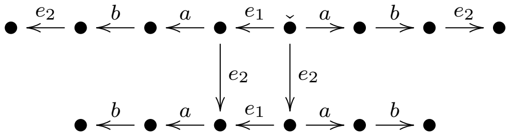
Example 3.8 This example illustrates an automaton that satisfies $DC1$, $DC2$ and $DC4$, but it is indecomposable as it does not fulfill $DC3$, since new strings appear in $\overset{3}{\underset{i=1}{||}} P_i(A_S)$ from the interleaving of strings in $P_1(A_S)$, $P_2(A_S)$ and $P_3(A_S)$, but they are not legal in A_S. Consider the task automaton A_S:



with $E_1 = \{a, b, e_1\}$, $E_2 = \{a, b, e_2\}$, $E_3 = \{b\}$, leading to $P_1(A_S) \cong$ 

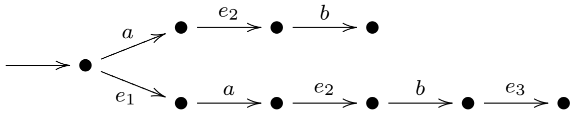
, $P_2(A_S) \cong$ 

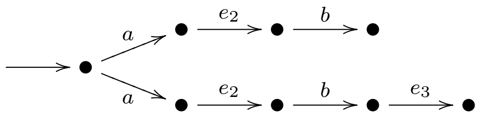
, $P_3(A_S) \cong$ 

and $\overset{3}{\underset{i=1}{||}} P_i(A_S)$: 

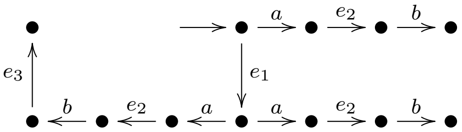
that is not bisimilar to A_S since two strings e_1abe_2 and e_2ab are newly generated, while they do not appear in A_S.

Example 3.9 This example illustrates an automaton that satisfies $DC1$ and $DC2$, and $DC3$, but is indecomposable as it does not fulfill $DC4$. Consider A_S:



with $E = E_1 \cup E_2 \cup E_3$, $E_1 = \{a, b, e_1, e_2, e_3\}$, $E_2 = E_3 = \{a, b, e_2, e_3\}$. Then, the parallel composition of $P_1(A_S) \cong A_S$, $P_2(A_S) \cong P_3(A_S) \cong$ 

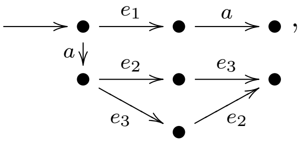
is $\overset{3}{\underset{i=1}{||}} P_i(A_S)$:

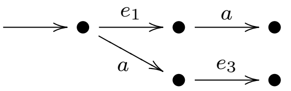


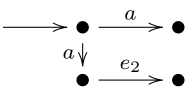
which is not bisimilar to A_S. The task automaton A_S satisfies $DC1$ and $DC2$ since any successive/adjacent pair of events belong to E_1. Furthermore, any interleaving in $\overset{3}{\underset{i=1}{||}} P_i(A_S)$ appears in A_S, and hence $DC3$ is satisfied; however, $DC4$ is violated as there does not exist deterministic automata $P_2'(A_S)$ and $P_3'(A_S)$ that respectively bisimulate $P_2(A_S)$ and $P_3(A_S)$. Therefore, $A_S \ncong \overset{3}{\underset{i=1}{||}} P_i(A_S)$, since, although $A_S \prec \overset{3}{\underset{i=1}{||}} P_i(A_S)$ and $\overset{3}{\underset{i=1}{||}} P_i(A_S) \prec A_S$, the simulation relations are not inverse of each other.

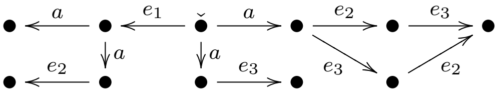
Remark 3.4 (Complexity) The complexity of the proposed method has the order of $|Q|^{n+1} \times |E|^3 \times log|Q|$; while the complexity of the method with constructing the parallel composition of the natural projections and checking the bisimilarity with the initial automaton is of the order of $|Q|^{2n} \times |E|$. Here, $|Q|$, $|E|$ and n denote the size of the state space, the size of the event set and the number of agents (number of local event sets), respectively. It can be seen that two methods cannot be compared generally in the sense that only when $|E| < \sqrt{|Q|^{n-1}/log|Q|}$, then the proposed method provides less complexity. In other words, the comparison of their complexity relies on the relative size of the event set with respect to the size of the state space.

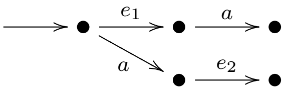
More importantly, the proposed method provides some guideline on the structure of the global specification automaton and the distribution the events among the agents in order for decomposability.

Remark 3.5 (Insights on Enforcing the Decomposability Conditions) The result in Theorem 3.2 or Corollary 3.2 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 $\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i$ nor $\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!$, 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 $E_i \in \{E_1, \ldots, E_n\}$ such that $\{e_1, e_2\} \subseteq E_i$. This solution also works for an indecomposability of A_S due to a violation of $DC2$ where $\exists e_1, e_2 \in E, q \in Q$, $s \in E^*$: $\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!$ but neither $\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i$ nor $\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!$. For instance, in Examples 3.5 and 3.6, if $E_2 = \{e_1, e_2\}$ and $E_3 = \{e_2, e_3\}$, then $DC1$ and $DC2$ were respectively satisfied. Violation of other two conditions, $DC3$ and $DC4$, is caused due to the 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 the 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 $\overset{n}{\underset{i=1}{||}} P_i\,(A_S)$ 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 $\overset{n}{\underset{i=1}{||}} P_i\,(A_S)$, then $DC4$ is violated. One way to remove this ambiguity is therefore to introduce 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 the identical strings from A_S. For example, the task automaton A_S: 

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

, and $P_2(A_S) \cong$ 

, is $P_1(A_S)||P_2(A_S) \cong$ 

$\ncong A_S$. Now, including e_1 in E_2 leads to $P_2(A_S) \cong$ 

and makes A_S decomposable. This issue will be discussed in details in Chapter 5 in order to enforce the decomposability conditions.

Remark 3.6 (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 "$s \in E^*$"; hence, it can be checked over the finite number of states and transitions. The second condition $DC2$: $\forall e_1, e_2 \in E, q \in Q, s \in E^*, \forall E_i \in \{E_1, ..., E_n\}, \{e_1, e_2\} \not\subset E_i$: $\delta(q, e_1 e_2 s)! \Leftrightarrow \delta(q, e_2 e_1 s)!$; (or $DC2$: $\forall e_1, e_2 \in E, q \in Q,\ s \in E^*$: $[\delta(q, e_1 e_2 s)! \vee \delta(q, e_2 e_1 s)!] \Rightarrow [\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i] \vee [\delta(q, e_1 e_2 s)! \wedge \delta(q, e_2 e_1 s)!]$) 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_4 in Example 2.2 violates $DC2$ as $(\delta(q_0, e_1 e_2), \delta(q_0, e_2 e_1)) \in \hat{R}_2$, $\exists e_2 \in E$, $\delta(\delta(q_0, e_1 e_2), e_2)!$, but $\neg\delta(\delta(q_0, e_2 e_1), e_2)!$.

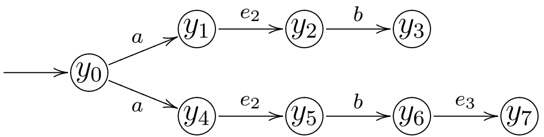
Checking $DC3$ also can be done over finite states by corresponding the strings s_i, $i = 1, \ldots, n$, in $DC3$ to the sequences $q_{1,i} \overset{e_{1,i}}{\to} q_{2,i} \overset{e_{2,i}}{\to} \ldots \overset{e_{n_i-1,i}}{\to} q_{n_i,i}$, $i = 1, \ldots, n$ with the respective path automata $A_i = (\{q_{1,i}, \ldots, q_{n_i,i}\}, \{q_{1,i}\}, \{e_{1,i}, \ldots, e_{n_i-1,i}\}, \xi_i)$, and checking that $\overset{n}{\underset{i=1}{\|}} P_i(A_i) \prec A_S$, where P_i is the natural projection into local event set E_i of the i-th agent. For example, the task automaton A_S in Remark 3.5 does not satisfy $DC3$ as, denoting its strings as $s_1 := e_1 a$, $s_2 := a e_2 e_3$ and $s_3 := a e_3 e_2$ with the corresponding path automata A_1:= $\longrightarrow \bullet \xrightarrow{e_1} \bullet \xrightarrow{a} \bullet$, A_2:= $\longrightarrow \bullet \xrightarrow{a} \bullet \xrightarrow{e_2} \bullet \xrightarrow{e_3} \bullet$, A_3:= $\longrightarrow \bullet \xrightarrow{a} \bullet \xrightarrow{e_3} \bullet \xrightarrow{e_2} \bullet$, it reveals that $P_1(A_1)||P_2(A_2)||P_3(A_1) \cong P_1(A_1)||P_2(A_3)||P_3(A_1)$: $\longrightarrow \bullet \xrightarrow{e_1} \bullet \xrightarrow{a} \bullet \xrightarrow{e_2} \bullet \nprec A_S$.

The fourth condition ($DC4$: $\forall i \in \{1, ..., n\}$, $x, x_1, x_2 \in Q_i$, $x_1 \neq x_2$, $e \in E_i$,

$t \in E_i^*$, $x_1 \in \delta_i(x, e)$, $x_2 \in \delta_i(x, e)$: $\delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$) 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. $x_4' \in \delta_i(x_4, e) \Rightarrow \exists x_3' \in Q_i$, $x_3' \in \delta_i(x_3, e)$, $(x_3', x_4') \in \hat{R}_4$.

For example, the task automaton in Example 3.9 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 $\neg\delta_2(y_3, e_3)!$.

Once the task is decomposed into local tasks and the local controllers are designed accordingly, similar to what we discussed in Section 3.2 for the hierarchical approach, the next and main question is the cooperative tasking problem (Problem 3.1) to understand whether the decomposable global task is guaranteed to be satisfied, provided the fulfilment of the local specifications. Following theorem provides a sufficient condition to answers Problem 3.1, using the decomposability result in Theorem 3.2.

Before stating the theorem, following lemma is presented to be used for the proof.

Lemma 3.6 *(Associativity of Parallel Composition [46])* $P_1(A_S) \parallel P_2(A_S) \parallel \cdots \parallel P_{n-1}(A_S) \parallel P_n(A_S) \cong P_n(A_S) \parallel (P_{n-1}(A_S) \parallel (\cdots \parallel (P_2(A_S) \parallel P_1(A_S))))$.

Theorem 3.3 *(Cooperative Tasking) Consider a plant, represented by a parallel dis-*

tributed system $\overset{n}{\underset{i=1}{\|}} A_{P_i}$*, with local event sets* E_i*,* $i = 1, ..., n$*, and let the global specification is given by a deterministic task automaton* A_S*, with* $E = \overset{n}{\underset{i=1}{\cup}} E_i$*. Then, designing local controllers* A_{C_i}*, so that* $A_{C_i} \parallel A_{P_i} \cong P_i(A_S)$*,* $i = 1, \cdots, n$*, derives the global closed loop system to satisfy the global specification* A_S*, in the sense of bisimilarity, i.e.,* $\overset{n}{\underset{i=1}{\|}} (A_{C_i} \parallel A_{P_i}) \cong A_S$*, provided* $DC1$*,* $DC2$*,* $DC3$ *and* $DC4$ *for* A_S*.*

Proof: Satisfying $DC1$-$DC4$ for A_S, according to Theorem 3.2, leads to the decomposability of A_S into local task automata $P_i(A_S)$, $i = 1, ..., n$, such that $A_S \cong \overset{n}{\underset{i=1}{\|}} P_i(A_S)$. Then, choosing local controllers A_{C_i}, so that $A_{C_i} \parallel A_{P_i} \cong P_i(A_S)$, $i = 1, 2, \cdots, n$, due to Lemma 2.6, results in $\overset{n}{\underset{i=1}{\|}} (A_{C_i} \parallel A_{P_i}) \cong \overset{n}{\underset{i=1}{\|}} P_i(A_S) \cong A_S$. ■

Now, if $DC1$-$DC4$ is reduced to $DC1$-$DC3$ (conditions in Theorem 3.2 are reduced into the conditions in Lemma 3.3), then $\overset{n}{\underset{i=1}{\|}} P_i(A_S) \cong A_S$ is reduced into $\overset{n}{\underset{i=1}{\|}} P_i(A_S) \prec A_S$; hence, choosing local controllers A_{C_i}, so that $A_{C_i} \parallel A_{P_i} \prec P_i(A_S)$, $i = 1, 2, \cdots, n$, due to Lemma 2.6 leads to $\overset{n}{\underset{i=1}{\|}} (A_{C_i} \parallel A_{P_i}) \prec \overset{n}{\underset{i=1}{\|}} P_i(A_S) \prec A_S$. Therefore,

Corollary 3.4 *Considering the plant and global task as stated in Theorem 3.3, if* $DC1$-$DC3$ *are satisfied, then designing local controllers* A_{C_i}*, so that* $A_{C_i} \parallel A_{P_i} \prec P_i(A_S)$*,* $i = 1, \cdots, n$*, derives the global closed loop system to satisfy the global specification* A_S*, in the sense of similarity, i.e.,* $\overset{n}{\underset{i=1}{\|}} (A_{C_i} \parallel A_{P_i}) \prec A_S$*.*

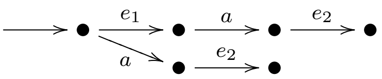
Remark 3.7 It is known that bisimulation implies language equivalence and that bisimulation of deterministic automata is reduced to their language equivalence. Now, one question is that whether for a deterministic task automaton its decomposability in the sense of bisimulation (stated in Theorem 3.2) is reduced to its decomposability in the sense of language equivalence ($L(A_S) = L(\overset{n}{\underset{i=1}{\parallel}} P_i\,(A_S))$) or its language separability ($L(A_S) = \overset{n}{\underset{i=1}{|}} L(P_i\,(A_S))$). Furthermore, it is interesting to know whether the proposed top-down cooperative control, in Theorem 3.3, is reduced into a top-down approach in the sense of language equivalence. As it is illustrated in the following examples, although in general, decomposability in the sense of bisimulation implies the decomposability in the sense of language equivalence, the reverse is not always true, in spite of the determinism of automaton. For the top-down cooperative control, on the other hand, under the proposed decomposability conditions, the bisimulation-based approach is reduced to the language equivalence one, as a deterministic task automaton can be represented by its language. As it was explained in Section 1.5, the advantage of using bisimulation for a deterministic A_S is that it allows to compare the behaviors of A_S and $\overset{n}{\underset{i=1}{\parallel}} P_i\,(A_S)$ using the sequential comparison of post-transitions of the corresponding states, by which less memory will be required rather than the comparison of strings.

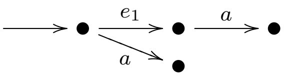
To elaborate these remarks, we first highlight that the natural projection may impose emerging properties that do not exist in the original automaton. For example, local task automata may have some new strings that do not appear in the original automaton, i.e., A_S does not necessarily simulates $P_i\,(A_S)$. Moreover, local task automata may become nondeterministic, even if the original task automaton is deterministic. The decomposability of A_S, however, concerns with the bisimilarity of A_S and $\overset{n}{\underset{i=1}{\parallel}} P_i\,(A_S)$, that may hold even if $P_i(A_S) \not\prec A_S$, or the local task automata are nondeterministic for some agents, as it is shown through the following examples.

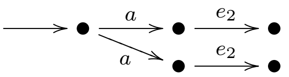
3.4.3 Examples for Remark 3.7

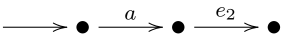
Example 3.10 Following example shows an automaton that does not simulate its natural projections, yet is decomposable. Consider the task automaton in Example 1.4. A_S is decomposable, although $P_1(A_S) \not\prec A_S$ (since the string b appears in $P_1(A_S)$, but not in A_S), and $P_2(A_S) \not\prec A_S$ (since the string ab appears in $P_2(A_S)$, but not in A_S).

As mentioned in Remark 3.7, another emergent property is that the natural projection of local task automata may lead to the nondeterminism of $P_i\left(A_S\right)$, leading to the nondeterminism of $\mathop{\|}\limits_{i=1}^{n} P_i\left(A_S\right)$. The decomposability of A_S again concerns with the bisimilarity of A_S and $\mathop{\|}\limits_{i=1}^{n} P_i\left(A_S\right)$, that may happen even if there exist some nondeterministic $P_i(A_S)$, as it is elaborated in the following example.

Example 3.11 Consider the automaton A_S: 

with $E = E_1 \cup E_2$, $E_1 = \{a, e_1\}$, $E_2 = \{a, e_2\}$. A_S is decomposable, as the parallel composition of $P_1(A_S)$: 

and $P_2(A_S)$: 

is bisimilar to A_S. Here, $P_2(A_S)$ is not deterministic, but it bisimulates the deterministic automaton $P_2(A_S)'$: 

.

Therefore, a deterministic task automaton A_S may have nondeterministic natural projections, and consequently, its $\mathop{\|}\limits_{i=1}^{n} P_i\left(A_S\right)$ may become nondeterministic. As a result, the determinism of A_S does not reduce its decomposability in the sense of bisimulation into its decomposability in the sense of language equivalence (synthesis modulo language equivalence [74]), due to the possibility of nondeterminism of $\mathop{\|}\limits_{i=1}^{n} P_i\left(A_S\right)$, as it is further illustrated in the following example.

Example 3.12 Consider the task automaton A_S in Examples 1.8 and 2.6, where A_S is deterministic, $L(\overset{n}{\underset{i=1}{\parallel}} P_i\,(A_S)) = L(A_S)$; however, $\overset{n}{\underset{i=1}{\parallel}} P_i\,(A_S) \not\cong A_S$.

This example also shows that the determinism of A_S also does not reduce its decomposability in the sense of bisimulation into the separability of its language ([77]), as $\overset{n}{\underset{i=1}{\parallel}} P_i\,(A_S) \not\cong A_S$, although A_S is deterministic and its language is separable $(L(A_S) = \overset{n}{\underset{i=1}{|}} L(P_i\,(A_S)))$. Another such automaton can be found in Example 3.9.

Therefore, in general for a deterministic task automaton $\overset{n}{\underset{i=1}{\parallel}} P_i\,(A_S) \cong A_S$ is not reduced into $L(A_S) = \overset{n}{\underset{i=1}{|}} L(P_i\,(A_S))$. But, under the determinism of bisimulation quotient of all local task automata ($DC4$), the bisimulation-based decomposability is reduced to the language-based decomposability and the top-down design based on bisimulation, is reduced to the language-based top-down design, such that the entire closed loop system (the parallel composition of local closed loop systems) bisimulates (or equivalently is language equivalent to) the global task automaton. In case of $DC4$, the other three conditions ($DC1$-$DC3$) can be used to characterize the language separability.

3.4.4 Special Case 1: Centralized Decision Making

This part introduces the centralized decision making on the selection ($DC1$) and order ($DC2$) of transitions as a special case of decentralized decision making. Accordingly, in the centralized decision making, the first two decomposability conditions are relaxed. By decentralized, here, we mean that there is no centralized agent for the coordination of all agents for global decision making; whereas, in centralized scheme there exists at least one agent that synchronizes all other agents on the order and switches between events. It is worth noting that this type of decentralized control has the communication constraint as the agents need the communication to synchronize on the common events. This means that the agents are not fully autonomous and need the communication, however, the agents sense locally and collaborate with their neighbors to achieve a global decision without global information and with no central unit. If at least one of the agents is equipped with global information and actuation then it will serve as a central unit for decision making, ruling the whole set of orders and selections, as it is stated in the following result.

Proposition 3.1 *Consider a deterministic automaton* $A_S = \left(Q, q_0, E = \bigcup\limits_{i=1}^{n} E_i, \delta\right)$. *If* $\exists E_k \in \{E_1, ..., E_n\}$, $E_k = E$, *then* $DC1$ *and* $DC2$ *always hold true.*

Proof: See the proof in the Appendix. ∎

In this case, $DC1$ and $DC2$ are relaxed in Theorem 3.2; hence, the decomposability result is reduced to

Corollary 3.5 *Consider a deterministic automaton* $A_S = \left(Q, q_0, E = \bigcup\limits_{i=1}^{n} E_i, \delta\right)$. *If* $\exists E_k \in \{E_1, ..., E_n\}$, $E_k = E$, *then* $A_S \cong \mathop{||}\limits_{i=1}^{n} P_i(A_S)$ *if and only if*

- *DC3:* $\delta(q_0, \mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)})!$ *for* $s_i \in \tilde{L}(A_S)$, *where,* $\tilde{L}(A_S) \subseteq L(A_S)$ *is the largest subset of* $L(A_S)$ *such that* $\forall s \in \tilde{L}(A_S), \exists s' \in \tilde{L}(A_S), \exists E_i, E_j \in \{E_1, ..., E_n\}, i \neq j, p_{E_i \cap E_j}(s)$ *and* $p_{E_i \cap E_j}(s')$ *start with the same event, and*
- *DC4:* $\forall i \in \{1, \cdots, n\}$, $x, x_1, x_2 \in Q_i$, $x_1 \neq x_2$, $e \in E_i$, $t \in E_i^*$, $x_1 \in \delta_i(x, e)$, $x_2 \in \delta_i(x, e)$: $\delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$.

It should be highlighted that the existence of a centralized decision maker does not necessarily imply the decomposability of the global task automaton. For instance, the automaton A_S in Example 1.8 is not decomposable, as it was discussed in Example 2.6, although $E_1 = E$.

3.4.5 Special Case 2: Mutual Exclusive Clusters of Local Event Sets

Generally, $DC3$ in Theorem 3.2 is more restrictive than its counterpart in Theorem 2.1 as it had excluded the identical strings s and s', while $DC3$ in Theorem 3.2 may check interleavings on identical strings. This means that $DC3$ in this case may check some of the interleavings that have been already checked by $DC1$ and $DC2$. For instance, in Example 3.3, one needs to only check $\delta(q_0, \overline{p_1(s_1)}|\overline{p_2(s_2)}|\overline{p_3(s_1)})!$ and $\delta(q_0, \overline{p_1(s_2)}|\overline{p_2(s_1)}|\overline{p_3(s_2)})!$ for $DC3$, the other interleaving transitions are redundant as they are also checked via $DC1$ and $DC2$ (all of them contain the successive projections of an identical string into different local event sets). Removing these overlapping will reduce the computational burden, however, it will impose complexity in the statements of $DC3$. For large scale systems it is difficult to determine the redundant interleavings for $DC3$. Moreover, the set of redundant interleavings depends on the problem. For instance, in Example 3.8, $\overline{p_1(s_1)}|\overline{p_2(s_2)}|\overline{p_3(s_2)}$ causes a violation of $DC3$, whereas in Example 3.3 it does not (s_1 and s_2 are the branches numbered from the top to bottom). Therefore, it is more tractable to let those transitions that are not defined over strings in $\tilde{L}(A_S)$ to be checked by $DC1$ and $DC2$, and the rest of transitions whose strings may interleave with other strings in $\tilde{L}(A_S)$, are checked by $DC3$.

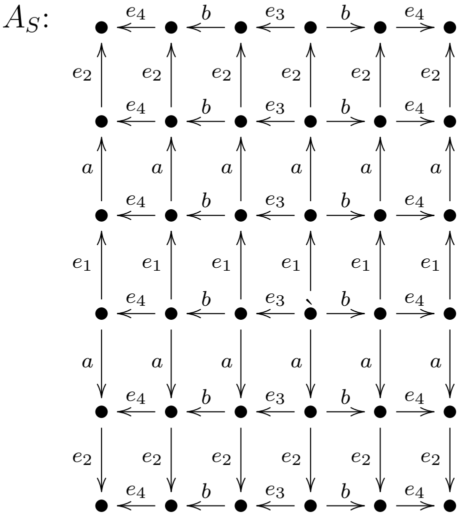
In special case that the event set is a partition of clusters of local event sets, the restrictiveness of $DC3$ is mitigated using the following result.

Proposition 3.2 *Let* $\{\Sigma^1, ..., \Sigma^K\}$ *be a partition of* E *such that* $E = \bigcup\limits_{k=1}^{K} \Sigma^k$, $\Sigma^i \cap \Sigma^j = \emptyset$, $\forall i, j \in \{1, ..., K\}, i \neq j$, $\Sigma^k = \bigcup\limits_{l=1}^{n_k} E_l^k$, $E_l^k \in \{E_1, ..., E_n\}$. *Then,* $A_S \cong \mathop{||}\limits_{i=1}^{n} P_i(A_S)$ *if and only if* $DC1$ *and* $DC2$ *hold true for* A_S *with respect to* $E_1, \ldots, E_n$, *and* $\forall k \in \{1, ..., K\} : P_{\Sigma^k}(A_S) \cong \mathop{||}\limits_{l=1}^{n_k} P_{E_l^k}(A_S)$.

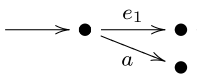
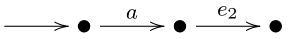
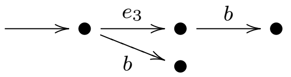
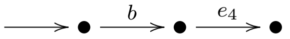
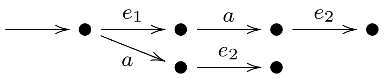
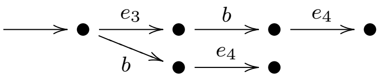
Proof: See the proof in the Appendix. ∎

The significance of this result is that if $DC1$ and $DC2$ hold true for A_S, then the decomposability of A_S is reduced to the decomposability of its projections $P_{\Sigma^k}(A_S)$ into the clusters, $\forall k \in \{1, ..., K\}$. This simplification is illustrated in the following example.

Example 3.13 Consider the global task automaton



with $E_1 = \{a, e_1\}$, $E_2 = \{a, e_2\}$, $E_3 = \{b, e_3\}$, $E_4 = \{b, e_4\}$. There are 70 strings that share the first appearing common events. Since only $p_1(s)$ is related to $p_2(s')$, and $p_3(s)$ is related to $p_4(s')$ for any strings s and s', totally, $70 \times 70 \times 2 = 9800$ combinations of interleaving transitions are required to be checked for $DC3$.

However, since $\Sigma_1 = (E_1 = \{a, e_1\}) \cup (E_2 = \{a, e_2\})$, $\Sigma_2 = (E_3 = \{b, e_3\}) \cup (E_4 = \{b, e_4\})$, $E = \Sigma_1 \cup \Sigma_2$, $\Sigma_1 \cap \Sigma_2 = \emptyset$, then using Proposition 3.2, the number of interleaving transitions to be checked for $DC3$ is remarkably reduced into 4 transitions $\delta_{\Sigma^1}\left(x_0, \overline{p_1(s_1)}|\overline{p_2(s_2)}\right)!$, $\delta_{\Sigma^1}\left(x_0, \overline{p_1(s_2)}|\overline{p_2(s_1)}\right)!$, $\delta_{\Sigma^2}\left(y_0, \overline{p_3(s_3)}|\overline{p_4(s_4)}\right)!$ and $\delta_{\Sigma^2}\left(y_0, \overline{p_3(s_4)}|\overline{p_4(s_3)}\right)!$, from the interleaving between $P_1(P_{\Sigma^1}(A_S))$: , $P_2(P_{\Sigma^1}(A_S)) \cong$ , and $P_3(P_{\Sigma^2}(A_S))$: , $P_4(P_{\Sigma^2}(A_S)) \cong$ . Here, s_1, s_2 and s_3, s_4 are respectively the top and bottom branches in $P_{\Sigma^1}(A_S)$:  and $P_{\Sigma^2}(A_S)$: .

3.5 Conclusion

The chapter proposed a method for automaton decomposability, applicable in the 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 chapter has the following contributions: firstly, the task decomposability approach in Chapter 2, has been extended into a sufficient condition for more than two agents, using a hierarchical algorithm. The algorithm, however, was shown to be sufficient only, and moreover, it depends on the order of agents to be chosen, in turn, for the hierarchical decomposition. Therefore, generalizing the results in Chapter 2, the next contribution was devoted to propose new necessary and sufficient conditions for the decomposability of an automaton with respect to parallel composition and natural projections into an arbitrary finite number of local event sets. The decomposability, here, is checked, and in case of decomposability, the decomposition is performed in one stage with no dependency on the order of local event sets. Next, to address the cooperative tasking problem, 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. Finally, a cooperative control scenario has been developed and implemented to illustrate the concepts of task decomposition and cooperative tasking, in this chapter.

For special case that the global event set is partitioned into some clusters of local event sets, such that the task automaton is decomposable from the perspective of each cluster, it was shown that the decomposability of the global task automaton is reduced into the global decision makings on the orders and selections between the transitions. Moreover, for the special case that one local event set contains all events, it was shown that the first two decomposability conditions, on decision making on the orders and selections, are relaxed.

Next questions on this topic include fault-tolerant task decomposition in spite of failure in some events, and the decomposabilizability of an indecomposable task automaton by modifying the event distribution, as will be addressed in Chapters 4 and 5, respectively.

3.6 Appendix

3.6.1 Proof for Lemma 3.1

The proof of this lemma comes from the restatement of the expression $\forall e_1 \in E_1 \backslash E_2, e_2 \in E_2 \backslash E_1$, equivalently, as $\forall e_1, e_2 \in E, \nexists E_i \in \{E_1, E_2\}, \{e_1, e_2\} \subseteq E_i$ or $\forall e_1, e_2 \in E, \forall E_i \in \{E_1, E_2\}, \{e_1, e_2\} \not\subset E_i$.

3.6.2 Proof for Lemma 3.2

From Lemma 2.1, stating that for a deterministic automaton $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$, $A_S \prec P_1(A_S)||P_2(A_S)$, it leads to $P_{\bigcup\limits_{i=m}^{n} E_i}(A_S) \prec P_m(A_S)||P_{\bigcup\limits_{i=m+1}^{n} E_i}(A_S)$, $m = 1, ..., n-1$, for $A_S = (Q, q_0, E = \bigcup\limits_{i=1}^{n} E_i, \delta)$. Therefore, $A_S \cong P_{\bigcup\limits_{i=1}^{n} E_i}(A_S) \prec P_1(A_S)||P_{\bigcup\limits_{i=2}^{n} E_i}(A_S) \prec P_1(A_S)||P_2(A_S)||P_{\bigcup\limits_{i=3}^{n} E_i}(A_S) \prec \cdots \prec \mathop{||}\limits_{i=1}^{n} P_i(A_S)$.

3.6.3 Proof for Lemma 3.3

Sufficiency: Consider the deterministic automaton $A_S = (Q, q_0, E, \delta)$. The set of transitions in $\mathop{||}\limits_{i=1}^{n} P_i(A_S) = (Z, z_0, E, \delta_{||})$ is defined as $T = \{(x_0^1, \cdots, x_0^n) \xrightarrow{\mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)}} (x_1, \cdots, x_n) \in \prod\limits_{i=1}^{n} Q_i\}$, where $(x_0^1, \cdots, x_0^n) \xrightarrow{\mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)}} (x_1, \cdots, x_n)$ in $\mathop{||}\limits_{i=1}^{n} P_i(A_S)$ is the interleaving of strings $x_0^i \xrightarrow{p_i(s_i)} x_i$ in $P_i(A_S)$, $i = 1, \cdots, n$ (the projections of $q_0 \xrightarrow{s_i} \delta(q_0, s_i)$ in A_S. 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 $\Gamma = \{\mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)} | s_i \in E^*, q_0 \xrightarrow{s_i} \delta(q_0, s_i)\}$, where, $\Gamma_1 = \{\mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)} \in \Gamma | s_1, \cdots, s_n \notin \tilde{L}(A), s_1 = \cdots = s_n\}$, $\Gamma_2 =$

$\{\mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)} \in \Gamma | \exists s_i, s_j \in \{s_1, \cdots, s_n\}, s_i \neq s_j, \forall s_i \in \{s_1, \cdots, s_n\}, s_i \notin \tilde{L}(A)\}$, $\Gamma_3 = \{\mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)} \in \Gamma | s_i \in \tilde{L}(A)\}$.

For the interleavings in Γ_1, $\forall z, z_1 \in Z$, $e \in E$, $z_1 \in \delta_{||}(z, e)$: $\exists q, q_1 \in Q$, $\delta(q, e) = q_1$ such that $\forall z[j] \in \{z[1], \cdots, z[n]\}$ (the $j-th$ component of z), $\exists l \in loc(e)$, $z[j] = [q]_l$. Similarly, $\forall e' \in E$, $z_2 \in Z$, $z_2 \in \delta_{||}(z_1, e)$: $\exists q_2 \in Q$, $\delta(q_1, e') = q_2$. Now, if $\nexists E_i \in \{E_1, \cdots, E_n\}$, $\{e, e'\} \in E_i$, then the definition of parallel composition will furthermore induce that $\exists z_3 \in Z$, $z_3 \in \delta_{||}(z, e')$, $z_2 \in \delta_{||}(z_3, e)$. 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 $\forall t \in E^*$, $\delta_{||}(z_2, t)!$: $\delta(q_2, t)!$ and $\delta(q_4, t)!$. Therefore, any path automaton in $\mathop{||}\limits_{i=1}^{n} P_i(A_S)$ is simulated by A_S; hence, $\delta(q_0, \mathop{|}\limits_{i=1}^{n} \overline{p_i(s)})!$ in A_S, $\forall s \in L(A_S)$.

For the interleavings in Γ_2, from the definition of Γ_2, it follows that for any set of s_i, $\delta(q_0, s_i)!$, $i \in \{1, \cdots, n\}$, two cases are possible for Γ_2:

Case 1: $\forall s, s' \in \{s_1, \cdots, s_n\}$, $\forall E_i, E_j \in \{E_1, \cdots, E_n\}$: $p_{E_i \cap E_j}(s) = \varepsilon$ and $p_{E_i \cap E_j}(s') = \varepsilon$. In this case, the projections of such strings s_i can be written as $p_i(s_i) = e^i_1, \cdots, e^i_{m_i}$, $i = 1, \cdots, n$. The interleaving of these projected strings leads to a grids of states and transitions in $\prod\limits_{i=1}^{n} \prod\limits_{j_i=0}^{m_i} x^i_{j_i}$ as $(x^{i_1}_{j_1}, \cdots, x^{i_n}_{j_n}) \xrightarrow{e^i_j} (y^{i_1}_{j_1}, \cdots, y^{i_n}_{j_n})$, with $y^{i_k}_{j_i} = \begin{cases} x^{i_k}_{j_{i+1}}, & \text{if } i = i_k, j = j_i + 1 \\ x^{i_k}_{j_i}, & \text{otherwise} \end{cases}$ $j_i = 0, 1, \cdots, m_i$, $i = 1, \cdots, n$, $i_k = 1, \cdots, n$, $k = 1, \cdots, n$. This grid of transitions simulate counterpart transitions in A_S, as $\forall s, s' \in \{s_1, \cdots, s_n\}$, for all two successive/adjacent events e^i_j and $e^{i'}_{j'}$, both orders exist in A_S, due to $DC1$ and $DC2$; hence, $\delta(q_{j_i, i_k}, e^k_j) = q'_{j_i, i_k}$, $j_i = 0, 1, \cdots, m_i$, $i = 1, \cdots, n$, $i_k = 1, \cdots, n$, $k = 1, \cdots, n$. Therefore, for any choice of s_i corresponding to Γ_2, $\delta(q_0, \mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)})!$ in A_S.

Case 2: $\exists s, s' \in \{s_1, \cdots, s_n\}$, $\exists E_i, E_j \in \{E_1, \cdots, E_n\}$: $p_{E_i \cap E_j}(s) \neq \varepsilon$ 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 $t_1 = e_1 \cdots e_m, t'_1 = e'_1 \cdots e'_m \notin (E_i \cap E_j)^*, \forall i, j \in \{1, \cdots, n\}, i \neq j$, $\exists i, j \in \{1, \cdots, n\}, i \neq j$, $a, b \in E_i \cap E_j$ and $t_2, t'_2 \in E^*$. Therefore, due to synchronization constraint, the interleaving of strings will not evolve from a and b onwards; 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 $\delta(q_0, \mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)})!$ in A_S.

Therefore, for all strings s_i corresponding to Γ_2, $\delta(q_0, \mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)})!$ in A_S. This is also true for transitions related to Γ_3, provided $DC3$. Consequently, if $DC1$, $DC2$ and $DC3$ are satisfied, for all $s_i \in \tilde{L}(A_S)$, $\delta(q_0, \mathop{|}\limits_{i=1}^{n} \overline{p_i(s_i)})!$, and the sufficiency is proven.

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

If $DC1$ is violated, then $\exists e_1, e_2 \in E$, $q \in Q$, $\nexists E_i \in \{E_1, \cdots, E_n\}$, $\{e_1, e_2\} \subseteq E_i$, $[\delta(q, e_1)! \wedge \delta(q, e_2)!] \wedge \neg[\delta(q, e_1 e_2)! \wedge \delta(q, e_2 e_1)!]$. 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, $\forall i \in loc(e_1), j \in loc(e_2)$. This in turn, from the definition of parallel composition leads to $\delta_{||}(([q]_1, \cdots, [q]_n), e_1)! \wedge \delta_{||}(([q]_1, \cdots, [q]_n), e_2)!$ and $\delta_{||}(([q]_1, \cdots, [q]_n), e_1 e_2)! \wedge \delta_{||}(([q]_1, \cdots, [q]_n), e_2 e_1)!$. This means that $\delta_{||}(([q]_1, \cdots, [q]_n), e_1 e_2)! \wedge \delta_{||}(([q]_1, \cdots, [q]_n), e_2 e_1)!$ in $\mathop{||}\limits_{i=1}^{n} P_i(A_S)$, but $\neg[\delta(q, e_1 e_2)! \wedge \delta(q, e_2 e_1)!]$ in A_S, i.e., $\mathop{||}\limits_{i=1}^{n} P_i(A_S) \not\prec A_S$ which contradicts with the hypothesis.

If $DC2$ is not satisfied, then $\exists e_1, e_2 \in E$, $q \in Q$, $\nexists E_i \in \{E_1, \cdots, E_n\}$, $\{e_1, e_2\} \subseteq E_i$, $s \in E^*$, $\neg[\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!]$, i.e., $[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!] \wedge \neg[\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$. The expression $[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!]$ from the definition of natural projection and Lemma 3.2, respectively implies that $\delta_{||}(([q]_1, \cdots, [q]_n), e_1e_2)! \wedge \delta_{||}(([q]_1, \cdots, [q]_n), e_2e_1)!$ and $\delta_{||}(([q]_1, \cdots, [q]_n), e_1e_2s)! \wedge \delta_{||}(([q]_1, \cdots, [q]_n), e_2e_1s)!$ in $\mathop{||}\limits_{i=1}^{n} P_i(A_S)$. This in turn leads to $\delta_{||}(([q]_1, \cdots, [q]_n), e_1e_2s)! \wedge \delta_{||}(([q]_1, \cdots, [q]_n), e_2e_1s)!$ in $\mathop{||}\limits_{i=1}^{n} P_i(A_S)$, but $\neg[\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$ in A_S, that contradicts with $\mathop{||}\limits_{i=1}^{n} P_i(A_S) \prec A_S$.

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

3.6.4 Proof for Lemma 3.4

Sufficiency: Following lemma is used together with Lemma 2.7 during the proof of Lemma 3.4. Firstly, let A_1 and A_2 be substituted by A_S and $\mathop{||}\limits_{i=1}^{n} P_i(A_S)$, respectively, in Lemma 2.7 . Then, the existence of $A'_1 = A'_S$ in Lemma 2.7 is characterized by the following lemma.

Lemma 3.7 *Consider a deterministic automaton A_S and its natural projections $P_i(A_S)$, $i = 1, \cdots, n$. Then, there exists a deterministic automaton A'_S such that $A'_S \cong \mathop{||}\limits_{i=1}^{n} P_i(A_S)$ if and only if there exist deterministic automata $P'_i(A_S)$ such that $P'_i(A_S) \cong P_i(A_S)$, $i = 1, \cdots, n$.*

Proof: Let $A_S = (Q, q_0, E = \overset{n}{\underset{i=1}{\cup}} E_i, \delta)$, $P_i(A_S) = (Q_i, q_0^i, E_i, \delta_i)$, $P_i'(A_S) = (Q_i', q_{0,i}', E_i, \delta_i')$, $i = 1, \cdots, n$, $\overset{n}{\underset{i=1}{||}} P_i(A_S) = (Z, z_0, E, \delta_{||})$, $\overset{n}{\underset{i=1}{||}} P_i'(A_S) = (Z', z_0', E, \delta_{||}')$. Then, the proof of Lemma 3.7 is presented as follows.

Sufficiency: The existence of deterministic automata $P_i'(A_S)$ such that $P_i'(A_S) \cong P_i(A_S)$, $i = 1, \cdots, n$ implies that δ_i', $i = 1, \cdots, n$ are functions, and consequently from the definition of parallel composition (Definition 1.9), $\delta_{||}'$ is a function; hence, $\overset{n}{\underset{i=1}{||}} P_i'(A_S)$ is deterministic. Moreover, from Lemma 2.6, $P_i'(A_S) \cong P_i(A_S)$, $i = 1, \cdots, n$ lead to $\overset{n}{\underset{i=1}{||}} P_i'(A_S) \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$, meaning that there exists a deterministic automaton $A_S' := \overset{n}{\underset{i=1}{||}} P_i'(A_S)$ such that $A_S' \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$.

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 $i = 1, 2, \cdots,$ or n, then there does not exist a deterministic automaton A_S' such that $A_S' \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$.

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 $\exists q, q_1, q_2 \in Q$, $e \in E_1$, $t_1, t_2 \in (E_2 \backslash E_1)^*$, $t \in E^*$, $\delta(q, t_1 e) = q_1$, $\delta(q, t_2 e) = q_2$, $\delta(q_1, t)!$, but $\nexists t' \in E^*$, $\delta(q_2, t')!$, such that $p_1(t) = p_1(t')$, that particularly leads to $\delta(q_1, t)! \wedge \neg\delta(q_2, t')!$. From $\delta(q_1, t)! \wedge \neg\delta(q_2, t)!$, the definition of natural projection, the definition of parallel composition and Lemma 3.2 it follows that $([q_1]_1, ([q_1]_2, \ldots, [q_1]_n)) \in \delta_{||}(([q]_1, ([q]_2, \ldots, [q]_n)), t_1 e)$, $([q_2]_1, ([q_1]_2, \ldots, [q_1]_n)) \in \delta_{||}(([q]_1, ([q]_2, \ldots, [q]_n)), t_1 e)$, $\delta_{||}(([q_1]_1, ([q_1]_2, \ldots, [q_1]_n)), t)!$, whereas $\neg\delta_{||}(([q_2]_1, ([q_1]_2, \ldots, [q_1]_n)), t)!$ in $\overset{n}{\underset{j=1}{||}} P_i(A_S)$, implying that there does not exist a deterministic automaton A_S' such that $A_S' \cong \overset{n}{\underset{j=1}{||}} P_i(A_S)$, and the necessity is followed. ∎

Now, Lemma 3.4 is proven as follows.

Sufficiency: $DC4$ implies that there exist deterministic automata $P_i'(A_S)$ such that $P_i'(A_S) \cong P_i(A_S)$, $i = 1, \cdots, n$. Then, from Lemmas 2.6 and 3.7, it follows, respectively, that $\overset{n}{\underset{i=1}{||}} P_i'(A_S) \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$, and that there exists a deterministic automaton $A_S' := \overset{n}{\underset{i=1}{||}} P_i'(A_S)$ such that $A_S' \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$ that due to Lemma 2.7, it results in $R_1^{-1} = R_2$.

Necessity: Let A_S be deterministic, $A_S \prec \overset{n}{\underset{i=1}{||}} P_i(A_S)$ with the simulation relation R_1 and $\overset{n}{\underset{i=1}{||}} P_i(A_S) \prec A_S$ with the simulation relation R_2, and assume by contradiction that $R_1^{-1} = R_2$, but $DC4$ is not satisfied. The violation of $DC4$ implies that for $\exists i \in \{1, \cdots, n\}$, there does not exists a deterministic automaton $P_i'(A_S)$ such that $P_i'(A_S) \cong P_i(A_S)$. Therefore, due to Lemma 3.7, there does not exist a deterministic automaton A_S' such that $A_S' \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$; hence, according to Lemma 2.7, it leads to $R_1^{-1} \neq R_2$ which is a contradiction.

3.6.5 Proof for Lemma 3.5

Denoting $p := \forall E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \not\subset E_i$, $q := [\delta(q, e_1)! \wedge \delta(q, e_2)!]$, and $r := [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$, the equivalence of two statements for $DC1$ is followed from $(p \wedge q) \Rightarrow r \equiv q \Rightarrow (\neg p \vee r)$ (since $(p \wedge q) \Rightarrow r \equiv \neg(p \wedge q) \vee r \equiv (\neg p \vee \neg q) \vee r \equiv \neg q \vee (\neg p \vee r) \equiv q \Rightarrow (\neg p \vee r)$).

For $DC2$, the expression $\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!$ is equivalent to $[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!] \Rightarrow [\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$, since for any expressions A and B, $A \Leftrightarrow$

$B \equiv (A \vee B) \Rightarrow (A \wedge B)$ (since $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (A \Leftarrow B) \equiv (\neg A \vee B) \wedge (\neg B \vee A) \equiv [(\neg A \vee B) \wedge \neg B] \vee [(\neg A \vee B) \wedge A] \equiv [(\neg A \wedge \neg B) \vee (B \wedge \neg B)] \vee [(\neg A \wedge A) \vee (B \wedge A)] \equiv [(\neg A \wedge \neg B) \vee \bot] \vee [\bot \vee (B \wedge A)] \equiv [\neg (A \vee B)] \vee [B \wedge A] \equiv (A \vee B) \Rightarrow (A \wedge B)$). This leads $DC2$ to $\forall e_1, e_2 \in E$, $\forall E_i \in \{E_1, ..., E_n\}$, $\{e_1, e_2\} \not\subset E_i$, $q \in Q$, $s \in E^*$: $\{[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!] \Rightarrow [\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]\}$. Now, taking $p := \forall E_i \in \{E_1, \dots, E_n\}, \{e_1, e_2\} \not\subset E_i$, $q := [\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!]$, and $r := [\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$, the equivalence of two statements for $DC2$ is followed similarly from $(p \wedge q) \Rightarrow r \equiv q \Rightarrow (\neg p \vee r)$.

3.6.6 Proof for Proposition 3.1

If $\exists E_k \in \{E_1, ..., E_n\}$, $E_k = E$ then $\forall e_1, e_2 \in E$: $\{e_1, e_2\} \subseteq E_k$, and the consequent parts of $DC1$ and $DC2$ become true in Corollary 3.2.

3.6.7 Proof for Proposition 3.2

Let $\left\{\Sigma^1, ..., \Sigma^K\right\}$ be a partition of E such that $E = \bigcup\limits_{k=1}^{K} \Sigma^k$, $\Sigma^i \cap \Sigma^j = \emptyset$, $\forall i, j \in \{1, ..., K\}, i \neq j$, $\Sigma^k = \bigcup\limits_{l=1}^{n_k} E_l^k$, $E_l^k \in \{E_1, ..., E_n\}$. Then, Proposition 3.2 is proven by the combination of following two lemmas.

Lemma 3.8 *If* $A_S \cong \mathop{||}\limits_{i=1}^{n} P_i(A_S)$, *then* $\forall k \in \{1, ..., K\} : P_{\Sigma^k}(A_S) \cong \mathop{||}\limits_{l=1}^{n_k} P_{E_l^k}(A_S)$.

Proof: Due to the associativity and commutativity of parallel composition, $A_S \cong \mathop{||}\limits_{i=1}^{n} P_i(A_S)$ leads to $A_S \cong \mathop{||}\limits_{l=1}^{n} P_l(A_S) \cong \mathop{||}\limits_{k=1}^{K} \left(\mathop{||}\limits_{l=1}^{n_k} P_{E_l^k}(A_S) \right)$. Consequently, $\forall k \in \{1, ..., K\} : P_{\Sigma^k}(A_S) \cong P_{\Sigma^k}\left(\mathop{||}\limits_{k=1}^{K} \left(\mathop{||}\limits_{l=1}^{n_k} P_{E_l^k}(A_S) \right) \right) \cong \mathop{||}\limits_{l=1}^{n_k} P_{E_l^k}(A_S)$, where, the first bisimilary comes from the definitions of natural projection and bisimulation, and the second bisimilarity is deduced from the fact that because of the partitioning of E by $\{\Sigma^1, ..., \Sigma^K\}$, each E_l^k, $l \in \{1, ..., n_k\}$ appears in only Σ^k, and all transitions in $\overset{n_r}{\underset{l=1}{||}} P_{E_l^r}(A_S)$, $r \neq k$, are replaced by empty transitions, in $P_{\Sigma^k}\left(\overset{K}{\underset{k=1}{||}}\left(\overset{n_k}{\underset{l=1}{||}} P_{E_l^k}(A_S)\right)\right)$, and according to the definition of parallel composition, none of the transitions in $\overset{n_k}{\underset{l=1}{||}} P_{E_l^k}(A_S)$ are disabled by any transition in $\overset{n_r}{\underset{l=1}{||}} P_{E_l^r}(A_S)$, $r \neq k$. ■

Lemma 3.9 *If* $\forall k \in \{1, ..., K\} : P_{\Sigma^k}(A_S) \cong \overset{n_k}{\underset{l=1}{||}} P_{E_l^k}(A_S)$*, and* $DC1$ *and* $DC2$ *hold true for* A_S *with respect to* $\{E_1, \cdots, E_n\}$*, then* $A_S \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$*.*

Proof: Firstly, according to the defined partitioning of E, $\forall E_i \in \{E_1, \cdots, E_n\}$, $\exists \Sigma^k \in \{\Sigma^1, \cdots, \Sigma^K\}$, $E_i \in \Sigma^k$. Therefore, the expression $[\exists E_i \in \{E_1, \cdots, E_n\}, \{e_1, e_2\} \subseteq E_i]$ in the consequent of $DC1$ and $DC2$ in Corollary 3.2, leads to the expression $[\exists \Sigma^k \in \{\Sigma^1, \cdots, \Sigma^K\}$, $E_i \in \Sigma^k$, $E_i \in \{E_1^k, \cdots, E_{n_k}^k\}$, $\{e_1, e_2\} \subseteq E_i \in \Sigma^k]$. Consequently, $DC1$ and $DC2$ for A_S with respect to $\{E_1, \cdots, E_n\}$ lead to $DC1$ and $DC2$ for A_S with respect to $\{\Sigma^1, \cdots, \Sigma^K\}$.

Moreover, $\Sigma^i \cap \Sigma^j = \emptyset$, $\forall i, j \in \{1, ..., K\}, i \neq j$ guarantees $DC3$ for A_S with respect to $\{\Sigma^1, \cdots, \Sigma^K\}$.

Furthermore, according to Theorem 3.2, $\forall k \in \{1, ..., K\} : P_{\Sigma^k}(A_S) \cong \overset{n_k}{\underset{l=1}{||}} P_{E_l^k}(A_S)$ implies $DC4$ for $P_{\Sigma^k}(A_S)$ with respect to $\{E_l^k\}_{l=1}^{n_k}$, leading to the existence of deterministic automata $P'_{E_l^k}(A_S)$ such that $P'_{E_l^k}(A_S) \cong P_{E_l^k}(A_S)$, $\forall k \in \{1, \cdots, K\}, l \in \{1, \cdots, n_k\}$, that due to Lemmas 3.7 and 2.6, there exists a deterministic automaton $P'_{\Sigma^k}(A_S) := \overset{n_k}{\underset{l=1}{||}} P'_{E_l^k}(A_S)$ such that $P'_{\Sigma^k}(A_S) \cong \overset{n_k}{\underset{l=1}{||}} P_{E_l^k}(A_S) \cong P_{\Sigma^k}(A_S)$; therefore, $DC4$ becomes true for A_S with respect to $\{\Sigma^1, \cdots, \Sigma^K\}$.

Therefore, $DC1$-$DC4$ will be satisfied for A_S with respect to $\{\Sigma^1, \cdots, \Sigma^K\}$, i.e., $A_S \cong \overset{K}{\underset{k=1}{||}} P_{\Sigma^k}(A_S)$, that because of $P_{\Sigma^k}(A_S) \cong \overset{n_k}{\underset{l=1}{||}} P_{E_l^k}(A_S)$, it results in $A_S \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$. ∎

Chapter 4

Reliable Cooperative Tasking

4.1 Introduction

Once a multi-agent system is designed, its safety becomes a crucial property across the agents in order to prevent the irrevocable consequences for the system and users. Failures on the other hand are usually unavoidable due to the large scale nature and complex interactions among the distributed agents. It is therefore very important to introduce some degree of redundancy into the design so as to achieve fault-tolerance [124]. Towards this end, this chapter represents a continuation of the works in Chapters 2 and 3, and deals with the robustness issues of the proposed top-down design approach with respect to event failures in the multi-agent systems. The main concern under failure is whether a previously decomposable task still can be achieved collectively by the agents. Please note that no global information on failures is assumed, and each agent is only aware of failures around itself and just trying to accomplish its previously assigned subtask (assuming that the global task is decomposable before failures, and subtasks are obtained, accordingly). An interesting question is whether these agents can achieve the original global task in spite of event failures. If not, we would like to ask under what conditions the global task could be robustly accomplished. This is actually the fault-tolerance issue of the top-down design, and the results provide designers hints on which events are fragile with respect to failures, and whether redundancies are needed for sharing of some events. It is desired to share as few number of events as possible through the communication links to reduce the bandwidth, and consequently, the cost of the design. The main objective of this chapter is to identify conditions on failed events under which a decomposable global task can still be achieved successfully between cooperative agents.

This work differs from diagnosability and isolation problems [125] whose interest is on the detection and identification of the type of faults. In this work the faults are known and the question is the tolerance of systems against the faults. It also differs from reliable supervisory control [126, 127] that seeks the minimal number of supervisors required for the correct functionality of the supervised systems. Another different problem is robust supervisory control [128] that considers the plant as a set of possible plants and designs supervisor applicable for the whole range of the plants.

This work is related to the fault-tolerant supervisory control that has been widely studied in the context of discrete event systems. For examples, [129] proposed switching to another supervisor after fault detection. In another work, [130], the author proposed to re-synthesize the supervisor upon the fault occurrence. A framework for fault-tolerant supervisory control has been proposed in [131] and further explored in [132] by enforcing given specifications for non-faulty and faulty parts of the plant to ensure that the plant recovers from any fault within a bounded delay, such that the recovered plant is equivalent to the non-faulty plant. In [133] a fault was modeled as an uncontrollable event, that its occurrence causes a faulty behavior. They provided a necessary and sufficient condition for the existence of supervisor under failures, based on controllability, observability and relative-closure, together with the notions of state-stability [134,135], and language-stability [136,137]. In [138], a fault recovery result has been proposed by introducing normal, transient and recovery modes, such that the language of the closed loop systems is equal to a given language of the normal mode. Most of these works however address the language specifications and deal with decentralized supervisory control with distributed supervisor and monolithic plant.

In this chapter, continuing the works in Chapters 2 and 3, it is firstly observed that a necessary condition for preserving the decomposability is that the failed events can only be shared events and could be those that are only received from the other agents or sent to others, redundantly. In other words, a necessary condition for failed events is that they are not produced by the sensors/actuators of the corresponding agents, and that the failed events are not sent to other agents, unless there exist some alternative agents to relay them. We call these events as passive events in the agent (see Definition 4.1). Passive events indeed refer to the shared events through redundant communication links. Based on this notation, it seems that the failure of passive events have no effect on decomposability, as they do not fail in the sender agents and the receiver is just no longer informed about those events. However, it will be shown that although the passivity of failed events is a necessary condition for preserving the decomposability, some additional conditions are required for the task automaton to remain decomposable. The intuitive reason is that when a shared event fails, the corresponding agent can no longer use its information as a part of decision making on the order or switch between transitions. Moreover, the failure should satisfy some criteria to ensure that after the failures, the parallel composition of local automata neither generates a new string that is not allowed in the global automaton, nor prevents a string that is allowed in the global task automaton. In particular, while the passivity of failed events is a necessary condition to preserve the decomposability, it is shown that for a deterministic task automaton that experiences failures on passive events, the task automaton remains decomposable if and only if any required decisions on the order/switch between any pair of events can be accomplished by at least one of the agents after failure; no illegal string is allowed and no legal string is prevented by the composition of local task automata, after the failure. It is furthermore shown that under the passivity of failed events together with the proposed conditions, a previously decomposable and satisfied task automaton can be still achieved by the team of agents.

The rest of the chapter is organized as follows. Sections 4.2 presents the main result on decomposability under event failures and introduces the necessary and sufficient conditions under which a decomposable task automaton remains decomposable in spite of event failures, followed by illustrative examples for each condition. Next, it is shown in Sections 4.3 that under the passivity and the proposed conditions, if a previously decomposable task automaton has been achieved globally by local controllers, it will remain satisfied, in spite of event failures. As a special case, Section 4.4 provides more insight on global decision making on the selections and orders of transitions, in a two-agent case. Finally, the chapter concludes with remarks and discussions in Section 4.5. The proofs of lemmas are given in the Appendix.

4.2 Task Decomposability Under Event Failures

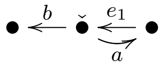
After identifying the conditions for task automaton decomposability and cooperative tasking, a natural follow-up question is that if after such decomposition, some of the events fail in some agents, then whether a previously decomposable and satisfied global task automaton can be still remain collectively satisfied after the event failures. And, if not, what are the conditions for preserving the cooperative tasking. In order to address this problem, we first need to investigate the failure on events. In general, an event e can be either private ($|loc(e)| = 1$) or shared ($|loc(e)| > 1$). Failure of a private event fails the decomposability as it causes the failure in the whole team of agents. Failure on a shared event, on the other hand, may or may not lead to a global failure, depending on whether the failed event is redundant or not. When an event is a sensor reading; or actuator command, or it is sent to other agents with no other alternative links, then the failure on this event stops its global evolutions. In the following, we will introduce a class of failures that are investigated in this work, followed by the class of passive failures.

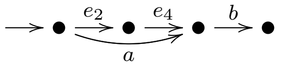
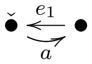
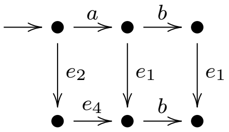
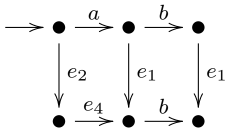
Definition 4.1 (Event Failure) Consider an automaton $A = (Q, q_0, E, \delta)$. An event $e \in E$ is said to be failed in A (or E), if $F(A) = P_\Sigma(A) = P_{E\backslash e}(A) = (Q, q_0, \Sigma = E\backslash e, \delta^F)$, where, Σ, δ^F and $F(A)$ denote the post-failure event set, post-failure transition relation and post-failure automaton, respectively. A set $\bar{E} \subseteq E$ of events is then said to be failed in A, when for $\forall e \in \bar{E}$, e is failed in A, i.e., $F(A) = P_\Sigma(A_i) = P_{E\backslash \bar{E}}(A) = (Q, q_0, \Sigma = E\backslash \bar{E}, \delta^F)$.

Consider a parallel distributed plant $A := \overset{n}{\underset{i=1}{||}} A_i = (Z, z_0, E = \overset{n}{\underset{i=1}{\cup}} E_i, \delta_{||})$ with local agents $A_i = (Q_i, q_0^i, E_i, \delta_i)$, $i = 1, \ldots, n$. The failure of e in E_i is said to be passive in E_i (or A_i) with respect to $\overset{n}{\underset{i=1}{||}} A_i$, if $E = \overset{n}{\underset{i=1}{\cup}} \Sigma_i$. An event whose failure in A_i is a passive failure is called a passive event in A_i.

The notion of passivity, can be interpreted as communication redundancy as for any local event set E_i, Σ_i excludes any passive failed event e from E_i, while the effect of this failure on $P_i(A_S)$ is defined as the projection of A_S into $E_i \backslash e$ (instead of E_i), leading to $P_{E_i \backslash e}(A_S)$. Moreover, the passivity of failed events is shown to be a necessary condition for the evolution of global transitions after failures, based on which the problems of decomposability and cooperative tasking under event failures are defined as follows.

Problem 4.1 *(Decomposability under Event Failures) Let a deterministic task automaton $A_S = (Q, q_0, E = \overset{n}{\underset{i=1}{\cup}} E_i, \delta)$ is decomposable with respect to parallel composition and natural projections P_i, $i = 1, \ldots, n$. Then, does the global task automaton A_S remain decomposable in spite of the failure of events $\{a_{i,r}\}$, $r \in \{1, ..., n_i\}$ in local event sets E_i, $i \in \{1, \ldots, n\}$? i.e., if $A_S \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$, then does $A_S \cong \overset{n}{\underset{i=1}{||}} F(P_i(A_S))$ always hold true? and if not, what are the conditions for such decomposability?*

Example 4.1 Consider the task automaton in Example 1.4 with $E = E_1 \cup E_2$ and local event sets $E_1 = \{a, b, e_1\}$, $E_2 = \{a, b, e_2, e_4\}$ and assume that the agent with E_2 sends two events $\{a, b\}$ to the agent with E_1. In this case, A_S is decomposable, since it bisimulates the parallel composition of $P_1(A_S)$:  and $P_2(A_S)$:

. Now, suppose that b fails to be sent to E_1, i.e., new E_1 becomes $\Sigma_1 = \{a, e_1\}$. In this case, the automaton A_S will no longer remain decomposable as the parallel composition of $F(P_1(A_S))$:  and $F(P_2(A_S)) \cong P_2(A_S)$, becomes $F(P_1(A_S))||F(P_2(A_S))$:  that is not bisimilar to A_S. If A_S was A_S: , then the failure of b in E_1 had no effect on the decomposability of A_S, i.e., $A_S \cong F(P_1(A_S))||F(P_2(A_S))$.

The next interesting question is the cooperative tasking under event failure, defined as

Problem 4.2 *(Cooperative Tasking under Event Failure) Consider a concurrent plant* $A_P := \overset{n}{\underset{i=1}{||}} A_{P_i}$ *and a decomposable deterministic task automaton* $A_S = (Q, q_0, E = \overset{n}{\underset{i=1}{\cup}} E_i, \delta) \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$*, and suppose that local controller automata* A_{C_i}*,* $i = 1, \ldots, n$ *exist such that each local closed loop system satisfies its corresponding local task, i.e.,* $A_{C_i}||A_{P_i} \cong P_i(A_S)$*,* $i = 1, \ldots, n$*. Assume furthermore that* $\bar{E}_i = \{a_{i,r}\}$ *fail in* E_i*,* $r \in \{1, ..., n_i\}$*. Then, does the team still can fulfill the global task, in spite of failures, i.e.,* $\overset{n}{\underset{i=1}{||}} F(A_{P_i}||A_{C_i}) \cong A_S$*? and if not, what are the conditions to preserve the satisfaction of the global specification?*

This and the following section respectively address the Problems 4.1 and 4.2 on automaton decomposability and cooperative tasking under event failures, for the class of passive failures. Firstly, according to the definition of passive events, the notion of passivity can be seen as the redundancy of communication links as it is stated as follows.

Remark 4.1 To interpret the passivity more formally, let $snd_e(i)$ and $rcv_e(i)$ respectively denote the set of labels that A_i sends e to those agents and the set of labels that A_i receives e from their agents, defined as $snd_e(i) = \{j \in \{1, ..., n\} | A_i \text{ sends } e \text{ to } A_j\}$ and $rcv_e(i) = \{j \in \{1, ..., n\} | i \in snd_e(j)\}$. Then, an event e is passive in A_i if $rec_e(i) \neq \emptyset$ (i.e., the $i-th$ agent does not receive e from its own sensor/actuator readings, but from another agent), and $\forall k \in snd_e(i)$: $\exists j \in \{1, \cdots, n\} \backslash \{i, k\}, k \in snd_e(j)$ (i.e., if the $i - th$ agent is a relay for the transmission of e, for any receiver agent, there exist another agent to send e). In this set-up a passive failure excludes the failed event e from the corresponding local event set E_i while it makes its respective transitions hidden in $F(A_i)$. Therefore, from the definition of parallel composition, the transitions on other agents can contribute to form the global transitions in $\overset{n}{\underset{i=1}{||}} F(A_i)$, since only in this way there will be no synchronization constraint on the rest of agents in $\overset{n}{\underset{i=1}{||}} F(A_i)$.

Moreover, the definition of passivity implies that the passivity of failed events is a necessary condition for the evolution of the global transitions after failures, as it is stated in the following lemma.

Lemma 4.1 *(Global Transitions after Local Failures) Consider a parallel distributed plant* $A := \overset{n}{\underset{i=1}{||}} A_i = (Z, z_0, E = \overset{n}{\underset{i=1}{\cup}} E_i, \delta_{||})$ *with local agents* $A_i = (Q_i, q_0^i, E_i, \delta_i)$*,* $i = 1, \ldots, n$*. If no global transitions in* $\overset{n}{\underset{i=1}{||}} A_i$ *are disabled in* $\overset{n}{\underset{i=1}{||}} F(A_i)$ *(i.e.,* $\forall z_1, z_2 \in Z$*,* $\forall e \in E$*,* $z_2 \in \delta_{||}(z_1, e)$*, then* $z_2 \in \delta_{||}^F(z_1, e)$*), then all event failures are passive, i.e., the passivity of local event failures is necessary for preserving the global transitions.*

Proof: See the proof in the Appendix. ∎

In this set-up, the evolution of global transitions in $\overset{n}{\underset{i=1}{||}} F(P_i\,(A_S))$ relies on the passivity of failed events, as it is expected and stated in Lemma 4.1. The reason is that due to the definition of parallel composition, the evolution of global transitions requires the failures to be passive, since passive failed events are excluded from the corresponding local event set and the local task automaton is projected to the rest of events. For non-passive failed events, on the other hand, since they are not received from other agents or they have no alternative agents for relaying the event; therefore, they are not excluded from the local event set, but their transitions are stopped. Consequently due to synchronization restriction in the definition of parallel composition, the global transitions cannot evolve on non-passive failed events.

Consequently, as it is stated in Lemma 4.1, the passivity of failed events is a necessary condition for the task automaton to remain decomposable after the failures.

Moreover, when all failed events are passive, due to the definition of passivity, Problem 4.1 can be transformed into the standard decomposition problem (Problem 2.1) to find the conditions under which $A_S \cong \overset{n}{\underset{i=1}{||}} P_{E_i\backslash\bar{E}_i}(A_S)$. Accordingly, the conditions on the global task automaton to preserve the decomposability under event failures, are reduced into their respective decomposability conditions in Corollary 3.2, as the following lemmas.

Lemma 4.2 *Consider a deterministic task automaton $A_S = (Q, q_0, E = \overset{n}{\underset{i=1}{\cup}} E_i, \delta)$. Assume that A_S is decomposable, i.e., $A_S \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$, and suppose that $\bar{E}_i = \{a_{i,r}\}$ fail in E_i, $r \in \{1, ..., n_i\}$, and $\bar{E}_i$ are passive for $i \in \{1, \ldots, n\}$. Then, following two* *expressions are equivalent:*

1. - *EF1:* $\forall e_1, e_2 \in E, q \in Q$*:* $[\delta(q, e_1)! \wedge \delta(q, e_2)!]$

     $\Rightarrow [\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i \backslash \bar{E}_i] \vee [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*

   - *EF2:* $\forall e_1, e_2 \in E, q \in Q,\ s \in E^*$*:* $[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!]$

     $\Rightarrow [\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i \backslash \bar{E}_i] \vee [\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$*.*

2. - *$DC1_\Sigma$:* $\forall e_1, e_2 \in E, q \in Q$*:* $[\delta(q, e_1)! \wedge \delta(q, e_2)!]$

     $\Rightarrow [\exists \Sigma_i \in \{\Sigma_1, \ldots, \Sigma_n\}, \{e_1, e_2\} \subseteq \Sigma_i] \vee [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$*;*

   - *$DC2_\Sigma$:* $\forall e_1, e_2 \in E, q \in Q,\ s \in E^*$*:* $[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!]$

     $\Rightarrow [\exists \Sigma_i \in \{\Sigma_1, \ldots, \Sigma_n\}, \{e_1, e_2\} \subseteq \Sigma_i] \vee [\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$*.*

Proof: See the proof in the Appendix. ∎

Lemma 4.2 gives the simplified versions of $DC1$ and $DC2$ after event failures, with respect to refined local event sets. Adopting the same $DC3$ for the refined local event sets, it remains to represent a simplified version of $DC4$ for the local task automata, after event failures. This condition is stated in the following lemma.

Lemma 4.3 *Consider a deterministic task automaton* $A_S = (Q, q_0, E = \overset{n}{\underset{i=1}{\cup}} E_i, \delta)$*. Assume that* A_S *is decomposable, i.e.,* $A_S \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$*, and suppose that* $\bar{E}_i = \{a_{i,r}\}$ *fail in* E_i*,* $r \in \{1, ..., n_i\}$*, and* $\bar{E}_i$ *are passive for* $i \in \{1, \ldots, n\}$*. Then, following two expressions are equivalent:*

- *EF4:* $\forall i \in \{1, \ldots, n\}$*,* $x, x_1, x_2 \in Q_i$*,* $x_1 \neq x_2$*,* $e \in E_i \backslash \bar{E}_i$*,* $t_1, t_2 \in \bar{E}_i^*$*,* $x_1 \in \delta_i(x, t_1e)$*,* $x_2 \in \delta_i(x, t_2e)$*:* $\delta_i(x_1, t_1')! \Leftrightarrow \delta_i(x_2, t_2')!$*, for some* t_1'*,* t_2' *such that* $p_{E_i \backslash \bar{E}_i}(t_1') = p_{E_i \backslash \bar{E}_i}(t_2')$*.*

- *$DC4_{\Sigma}$: $\forall i \in \{1, \ldots, n\}$, $x, x_1, x_2 \in Q_i$, $x_1 \neq x_2$, $e \in \Sigma_i$, $t \in \Sigma_i^*$, $x_1 \in \delta_i^F(x, e)$, $x_2 \in \delta_i^F(x, e)$: $\delta_i^F(x_1, t)! \Leftrightarrow \delta_i^F(x_2, t)!$. Where, δ_i^F is the transition relation in $F(P_i(A_S))$.*

Proof: See the proof in the Appendix. ■

Remark 4.2 $EF4$ is the counterpart of $DC4$ after the event failures, that handle newly possible nondeterminism in the local task automata. Any nondeterminism that is propagated from the local task automata of before the failure, is treated by $DC4$ when A_S is decomposable.

Now, combination of Corollary 3.2 and Lemmas 4.2 and 4.3 leads to the main result on decomposability under event failures as the following theorem.

Theorem 4.1 *(Task Decomposability under Event Failures) Consider a deterministic task automaton $A_S = (Q, q_0, E = \overset{n}{\underset{i=1}{\cup}} E_i, \delta)$. Assume that A_S is decomposable, i.e., $A_S \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$, and furthermore, assume that $\bar{E}_i = \{a_{i,r}\}$ fail in E_i, $r \in \{1, ..., n_i\}$, and $\bar{E}_i$ are passive for $i \in \{1, \ldots, n\}$. Then, A_S remains decomposable, in spite of event failures, i.e., $A_S \cong \overset{n}{\underset{i=1}{||}} F(P_i(A_S))$ if and only if*

- *$EF1$: $\forall e_1, e_2 \in E, q \in Q$: $[\delta(q, e_1)! \wedge \delta(q, e_2)!]$*

  $\Rightarrow [\exists E_i \in \{E_1, \cdots, E_n\}, \{e_1, e_2\} \subseteq E_i \backslash \bar{E}_i] \vee [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$;

- *$EF2$: $\forall e_1, e_2 \in E, q \in Q$, $s \in E^*$: $[\delta(q, e_1e_2s)! \vee \delta(q, e_2e_1s)!]$*

  $\Rightarrow [\exists E_i \in \{E_1, \cdots, E_n\}, \{e_1, e_2\} \subseteq E_i \backslash \bar{E}_i] \vee [\delta(q, e_1e_2s)! \wedge \delta(q, e_2e_1s)!]$;

- *$EF3$: $\delta(q_0, \overset{n}{\underset{i=1}{|}} \overline{p_i\left(s_i\right)})!$, $\forall \{s_1, \cdots, s_n\} \in \hat{L}\left(A_S\right)$, $\exists s_i, s_j \in \{s_1, \cdots, s_n\}, s_i \neq s_j$, where $\hat{L}\left(A_S\right) \subseteq L\left(A_S\right)$ is the largest subset of $L\left(A_S\right)$ such that $\forall s \in$*

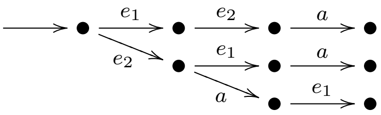
$\hat{L}(A_S), \exists s' \in \hat{L}(A_S)$, $\exists \Sigma_i, \Sigma_j \in \{\Sigma_1, ..., \Sigma_n\}, i \neq j, p_{\Sigma_i \cap \Sigma_j}(s)$ and $p_{\Sigma_i \cap \Sigma_j}(s')$ start with the same event, and

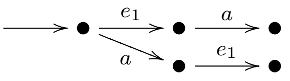
- *$EF4$: $\forall i \in \{1, \ldots, n\}$, $x, x_1, x_2 \in Q_i$, $x_1 \neq x_2$, $e \in E_i \backslash \bar{E}_i$, $t_1, t_2 \in \bar{E}_i^*$, $x_1 \in \delta_i(x, t_1 e)$, $x_2 \in \delta_i(x, t_2 e)$: $\delta_i(x_1, t_1')! \Leftrightarrow \delta_i(x_2, t_2')!$, for some t_1', t_2' such that $p_{E_i \backslash \bar{E}_i}(t_1') = p_{E_i \backslash \bar{E}_i}(t_2')$.*

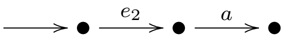
Proof: First, according to Lemma 4.1, the passivity of $\bar{E}_i$ is a necessary condition for preserving the decomposability. Now, providing the decomposability of A_S and the passivity of all failed events, due to the definition of passivity, it leads to $\overset{n}{\underset{i=1}{||}} F(P_i(A_S)) \cong \overset{n}{\underset{i=1}{||}} P_{\Sigma_i}(A_S) = \overset{n}{\underset{i=1}{||}} P_{E_i \backslash \bar{E}_i}(A_S)$ that based on Corollary 3.2 and Lemmas 4.2 and 4.3, it is bisimilar to A_S if and only if $EF1$ - $EF4$ hold true for the refined local event sets $\{\Sigma_1, \ldots, \Sigma_n\}$. ■

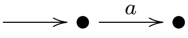
Remark 4.3 $EF1$-$EF4$ are respectively the decomposability conditions $DC1$-$DC4$, after event failures with respect to parallel composition and natural projections into refined local event sets $\Sigma_i = E_i \backslash \bar{E}_i$, $i \in \{1, \ldots, n\}$, provided the passivity of $\bar{E}_i$, $i \in \{1, \ldots, n\}$. Condition $EF1$ means that, after failure of some passive events, for any decision on 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). $EF2$ says that, after failure of some passive events, 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 the occurrence of that string. The condition $EF3$ means that, after failure of some passive events, any interleaving of strings from local task automata that have the same first appearing shared event, should not allow a string that is not allowed in the original task automaton. In other words, $EF3$ is to ensure that, after failure of some passive events, an illegal behavior (a string that does not appear in A_S) is not allowed by the team (does not appear in $\overset{n}{\underset{i=1}{||}} F(P_i\,(A_S))$). The last condition, $EF4$, ensures the determinism of bisimulation quotient of local task automaton, in order to guarantee the symmetry between simulation relations from A_S to $\overset{n}{\underset{i=1}{||}} F(P_i(A_S))$ and vice versa. By providing this symmetry property, $EF4$ guarantees that, after the failures, a legal behavior (a string in A_S) is not disabled by the team (appears in $\overset{n}{\underset{i=1}{||}} F(P_i(A_S))$.

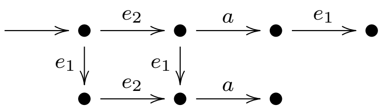
Following examples illustrate these conditions.

Example 4.2 This example illustrates the notion of passivity and shows a decomposable automaton that stays decomposable, when an event is failed passively in one of the local agents and $EF1$-$EF4$ are satisfied. Consider the automaton A_S: 

with local event sets $E_1 = \{e_1, a\}$ and $E_2 = \{e_2, a\}$, $E_3 = \{a\}$ and communication pattern as $\{1,2\} \in snd_a(3)$, and no other communication links. This automaton is decomposable, as the parallel composition of $P_1(A_S) \cong$ 

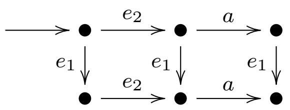
, $P_2(A_S) \cong$ 

and $P_3(A_S) \cong$ 

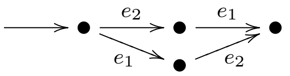
is $\overset{3}{\underset{i=1}{||}} P_i(A_S)$: 

which is bisimilar to A_S. Now, assume that a fails in E_1. Then $EF1$-$EF4$ are satisfied (as $\delta(q_0, e_2e_1a)! \wedge \delta(q_0, e_2ae_1)!$; hence, $EF1$ and $EF2$ hold true after the failure, the interleavings on shared event a impose no illegal strings, and therefore,

$EF3$ is satisfied, and finally $EF4$ is fulfilled since $F(P_1(A_S)) \cong \longrightarrow \bullet \xrightarrow{e_1} \bullet$, $F(P_2(A_S)) \cong \longrightarrow \bullet \xrightarrow{e_2} \bullet \xrightarrow{a} \bullet$ and $F(P_3(A_S)) \cong \longrightarrow \bullet \xrightarrow{a} \bullet$ are all deterministic); therefore, the parallel composition of $F(P_1(A_S))$ with $\Sigma_1 = \{e_1\}$, $F(P_2(A_S))$ with $\Sigma_2 = \{e_2, a\}$, and $F(P_3(A_S))$ with $\Sigma_3 = \{a\}$, is $\overset{3}{\underset{i=1}{||}} F(P_i(A_S))$:



that is bisimilar to A_S. However, if a was failed in E_3, then it evolved in none of the local task automata and $\overset{3}{\underset{i=1}{||}} F(P_i(A_S)) \ncong A_S$, since E_3 is the source for a. Similarly, the failure of private events e_1 and e_2 in E_1 and E_2, respectively, disables the global transitions on these events. As another example for non-passive failure, consider the communication pattern of $1 \in snd_a(3)$, $\{2,3\} \subseteq snd_a(1)$, while a fails in E_1, Then, the parallel composition of $F(P_1(A_S))$: $\longrightarrow \bullet \xrightarrow{e_1} \bullet$ with $\Sigma_1 = \{e_1, a\}$, $F(P_2(A_S)) \cong \longrightarrow \bullet \xrightarrow{e_2} \bullet$ with $\Sigma_2 = \{e_2\}$, and $F(P_3(A_S)) \cong \longrightarrow \bullet \xrightarrow{a} \bullet$ with $\Sigma_3 = \{a\}$ was



which is not bisimilar to A_S. The reason is that in this case, in contrast to the fist case, a was not excluded from Σ_1, while a was stopped in $F(P_1(A_S))$. This, due to the synchronization constraint in parallel composition, disabled the global transitions on a.

Example 4.3 (Revisiting Example 3.4 for reliable task decomposability) As another example, consider the multi-robot cooperative scenario in Section 3.3 and Examples 1.10 and 3.4, with $E_1 = \{h_1, R_1toD_1, R_1onD_1, FWD, D_1opened, R_2in1, BWD, D_1closed, r\}$, $E_2 = \{h_2, R_2to2, R_2in2, D_1opened, R_2to1, R_2in1, r\}$, 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 the communication protocol $\{1,3\} \in snd_{\{R_2in1\}}(2)$, $\{1,2\} \in snd_{\{r, D_1closed\}}(3)$, $\{2,3\} \in snd_{\{D_1opened\}}(1)$, $1 \in snd_{\{FWD, BWD\}}(3)$, $3 \in$

$snd_{\{FWD,BWD\}}(1)$. With this communication pattern, $D_1opened$, is passive in E_2 and E_3, $D_1closed$ and r are passive in E_1 and E_2, and R_2in1 is passive in E_1 and E_3. Moreover, D_1opend and R_2in1 are redundant in E_3, while $D_1closed$ is redundant in E_1. The reason is that $D_1opended$ is passive in E_3 and its exclusion respects $EF1$-$EF4$, as E_3 has no role in decision making on successive events $\{R_2in2, D_1opened\} \subseteq E_2$, $\{D_1opened, R_2to1\} \subseteq E_2$ that can be handled by E_2, and the decision on the order of $\{FWD, D_1opened\} \subseteq E_1$ that can be accomplished by E_1. Similarly, after the exclusion of passive event R_2in1 from E_3, the order of $\{R_2to1, R_2in1\} \subseteq E_2$, and $\{R_2in1, BWD\} \subseteq E_1$ can be decided by E_2 and E_1, respectively. Identically, $D_1closed$ is redundant in E_1, as it is passive and after its exclusion, E_3 can handle the order of event pairs $\{BWD, D_1closed\} \subseteq E_3$ and $\{D_1closed, R_3to1\} \subseteq E_3$. Therefore, $EF2$ remains satisfied. Moreover, since these events have no adjacent event; the shared events appear in only one branch, and local automata remain deterministic, $EF1$, $EF3$ and $EF4$ are also fulfilled. Consequently, the exclusion of $D_1opened$ and R_2in1 from E_3 and $D_1closed$ from E_1 preserve the decomposability of A_S with respect to the new local event sets $\Sigma_1 = \{h_1, R_1toD_1, R_1onD_1, FWD, D_1opened, R_2in1, BWD, r\}$, $\Sigma_2 = \{h_2, R_2to2, R_2in2, D_1opened, R_2to1, R_2in1, r\}$ and $\Sigma_3 = \{h_3,\ R_3to3,\ R_3in3,\ R_3toD_1,\ R_3onD_1,\ FWD,\ BWD,\ D_1closed,\ R_3to1,\ R_3in1,\ r\}$, as the parallel composition of $F(P_1(A_S))$, $F(P_2(A_S))$ and $F(P_3(A_S))$ (shown in Figure 4.1) bisimulates A_S.

Example 4.4 This example shows a decomposable automaton that will no longer stay decomposable after a passive event failure, since $EF1$ is not satisfied, although

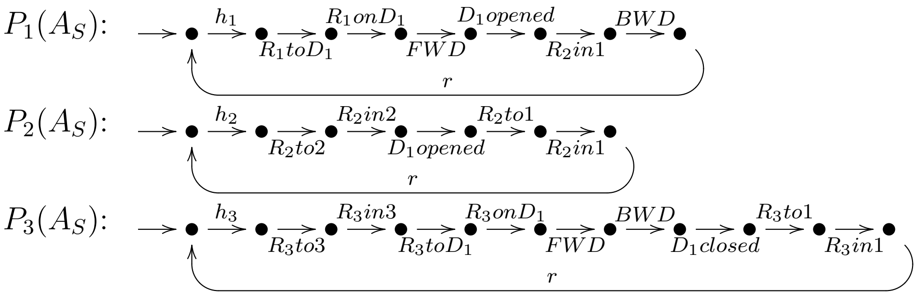
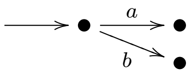
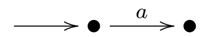
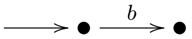
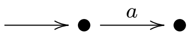


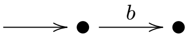
Figure 4.1: $F(P_1(A_S))$ for R_1; $F(P_2(A_S))$ for R_2 and $F(P_3(A_S))$ for R_3, after exclusion of D_1*closed* from E_1 and $\{D_1opened, R_2in1\}$ from E_3.

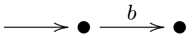
other three conditions, $EF2$, $EF3$ and $EF4$, are fulfilled. Consider the automaton A_S: 

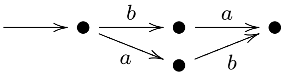
with local event sets $E_1 = \{a\}$, $E_2 = \{b\}$, and $E_3 = \{a, b\}$ with $3 \in snd_a(1)$ and $3 \in snd_b(2)$, and no other sending and receiving links. This automaton is decomposable, as the parallel composition of $P_1(A_S)$: 

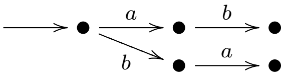
, $P_2(A_S)$: 

and $P_3(A_S) \cong A_S$ bisimulates A_S. Now, suppose that a is failed in E_3. Then, the parallel composition of $F(P_1(A_S))$: 

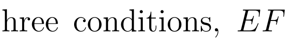
with $\Sigma_1 = \{a\}$, $F(P_2(A_S))$: 

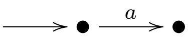
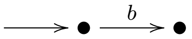
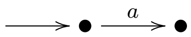
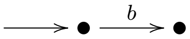
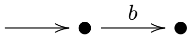
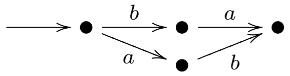
with $\Sigma_2 = \{b\}$, and $F(P_3(A_S))$: 

with $\Sigma_3 = \{b\}$, is $\overset{3}{\underset{i=1}{||}} F(P_i(A_S))$: 

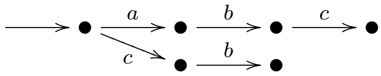
which is not bisimilar to A_S. The reason is the violation of $EF1$, as after the failure of a in E_3, neither there exists an agent that knows both events a and b to decide on the selection between them, nor both permutations are legal in A_S. If A_S was A_S: 

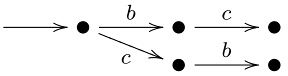
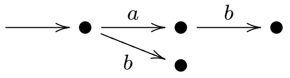
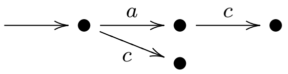
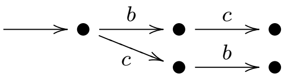
, then, the failure of a in E_3 had no effect on the decomposability of A_S.

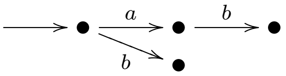
Example 4.5 This example shows a decomposable automaton that will no longer stay decomposable after a passive failure, as $EF2$ is not satisfied, although other three conditions, $EF1$, $EF3$ and $EF4$ are fulfilled. Consider the automaton A_S: 

with local event sets $E_1 = \{a\}$, $E_2 = \{b\}$ and $E_3 = \{a, b\}$, with $3 \in snd_a(1)$ and $3 \in snd_b(2)$ with no other sending and receiving links. This automaton is decomposable, as the parallel composition of $P_1(A_S)$: , $P_2(A_S)$:  and $P_3(A_S) \cong A_S$ bisimulates A_S. Now, suppose that a is failed in E_3. Then, the parallel composition of $F(P_1(A_S))$:  with $\Sigma_1 = \{a\}$, $F(P_2(A_S))$:  with $\Sigma_2 = \{b\}$, and $F(P_3(A_S))$:  with $\Sigma_3 = \{b\}$, is $\overset{3}{\underset{i=1}{||}} F(P_i(A_S))$:  which is not bisimilar to A_S. The reason is the violation of $EF2$, as after the failure of a in E_3, neither there exists an agent that knows both events a and b to decide on the order of them, nor both orders are legal in A_S.

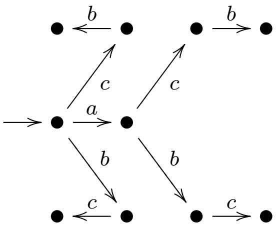
Example 4.6 This example illustrates a decomposable automaton that satisfies $EF1$, $EF2$ and $EF4$, but it will not remain decomposable after a passive event failure, due to the violation of $EF3$. Consider the automaton A_S:



with local event sets $E_1 = \{a, b, c\}$, $E_2 = \{b, c\}$ and $E_3 = \{a, b\}$ and communication pattern $1 \in snd_{\{b,c\}}(2)$, $1 \in snd_a(3)$, $3 \in snd_b(2)$, with no other communication links. A_S is decomposable, as the parallel composition of $P_1(A_S) \cong A_S$, $P_2(A_S)$: , and $P_3(A_S)$:  is bisimilar to A_S. Now, assume that b fails in E_1. Then, the parallel composition of $F(P_1(A_S))$:  with $\Sigma_1 = \{a, c\}$, $F(P_2(A_S))$:  with $\Sigma_2 = \{b, c\}$ and $F(P_3(A_S))$:

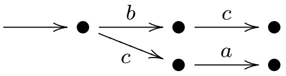


with $\Sigma_3 = \{a, b\}$ is $\overset{3}{\underset{i=1}{||}} F(P_i(A_S))$:

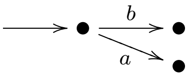


that is no longer bisimilar to A_S due to the violation of $EF3$ as it contains strings acb and bc that do not appear in A_S.

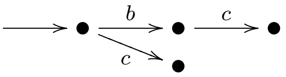
Example 4.7 This example shows a decomposable automaton that does not remain decomposable against a passive event failure, when it does not satisfy $EF4$, although it fulfils $EF1$, $EF2$ and $EF3$. Consider the automaton A_S:



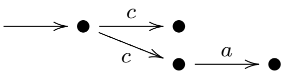
with local event sets $E_1 = \{a, b, c\}$, $E_2 = \{a, b\}$ and $E_3 = \{b, c\}$, with communication structure $1 \in snd_{\{a,b\}}(2)$, $1 \in snd_c(3)$, $3 \in snd_b(2)$, with no other communication links. This automaton is decomposable, as the parallel composition of $P_1(A_S) \cong A_S$, $P_2(A_S)$:



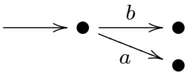
and $P_3(A_S)$:



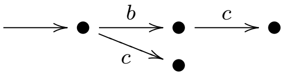
is bisimilar to A_S. Now, assume that b fails in E_1, then the parallel composition of $F(P_1(A_S))$:



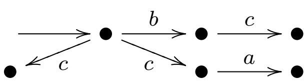
with $\Sigma_1 = \{a, c\}$, $F(P_2(A_S)) \cong$



with $\Sigma_2 = \{a, b\}$ and $F(P_3(A_S)) \cong$



with $\Sigma_3 = \{b, c\}$ is $\overset{3}{\underset{i=1}{||}} F(P_i(A_S))$:



that is no longer bisimilar to A_S due to the violation of $EF4$, as there does not exist a deterministic automaton $P_1'(A_S)$ such that $P_1'(A_S) \cong F(P_1(A_S))$.

Remark 4.4 The complexity and checkability of $EF1$-$EF4$ are similar to the complexity and decidability of $DC1$-$DC4$, as they were discussed in Remarks 3.4 and 3.6.

4.3 Cooperative Tasking Under Event Failure

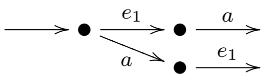
So far, we have presented the necessary and sufficient conditions for a decomposable task automaton to remain decomposable in spite of passive failures. Now, assume that the global task automaton is decomposable and local controllers exist such that local specifications are satisfied, and consequestly due to Theorem 3.3, the global specification is satisfied, by the team. Furthermore, assume that event failures occur on some shared events, but due to the passivity of failed events and $EF1$-$EF4$, the global task automaton remains decomposable. Then, the next question is Problem 4.2 to understand whether, the team is still able to achieve the global specification. Following result answers this question.

Theorem 4.2 *(Cooperative Tasking under Event Failures) Consider a concurrent plant* $A_P := \overset{n}{\underset{i=1}{||}} A_{P_i}$ *and a deterministic task automaton* $A_S = (Q, q_0, E = \overset{n}{\underset{i=1}{\cup}} E_i, \delta)$ *as the global specification. Assume that* A_S *is decomposable, i.e.,* $A_S \cong \overset{n}{\underset{i=1}{||}} P_i(A_S)$*, and suppose that the local controller automata* A_{C_i}*,* $i = 1, \ldots, n$*, exist such that each local closed loop system satisfies its corresponding local task, i.e.,* $A_{C_i}||A_{P_i} \cong P_i(A_S)$*,* $i = 1, \ldots, n$*. Assume furthermore that* $\bar{E}_i = \{a_{i,r}\}$ *fail in* E_i*,* $r \in \{1, ..., n_i\}$*,* $\bar{E}_i$ *are passive for* $i \in \{1, \ldots, n\}$*, and* A_S *satisfies* $EF1$-$EF4$*. Then, the team can still achieve its global specification, i.e.,* $\overset{n}{\underset{i=1}{||}} F(A_{P_i}||A_{C_i}) \cong A_S$*.*

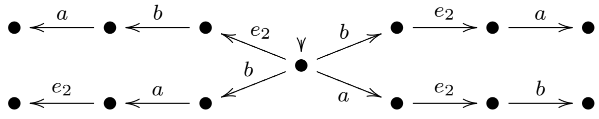
Proof: Firstly, the decomposability of A_S and $A_{C_i}||A_{P_i} \cong P_i(A_S)$, $i = 1, \ldots, n$, due to Theorem 3.3, implies that $\overset{n}{\underset{i=1}{||}} (A_{P_i}||A_{C_i}) \cong A_S$, i.e., the global specification is satisfied by the team. Moreover, the global specification remains satisfied, in spite of event failures, if $\bar{E}_i$ are passive for $i \in \{1, \ldots, n\}$, and A_S satisfies $EF1$-$EF4$, since $\overset{n}{\underset{i=1}{||}} F(A_{P_i}||A_{C_i}) \cong \overset{n}{\underset{i=1}{||}} P_{E_i \backslash \bar{E}_i}(A_{P_i}||A_{C_i}) \cong \overset{n}{\underset{i=1}{||}} P_{E_i \backslash \bar{E}_i}(P_i(A_S)) \cong \overset{n}{\underset{i=1}{||}} F(P_i(A_S)) \cong \overset{n}{\underset{i=1}{||}} P_i(A_S) \cong A_S$. In this expression, the first and the third bisimilarities come from the passivity of $\bar{E}_i$, $i \in \{1, \ldots, n\}$, and the second bisimilarity is followed from $A_{C_i}||A_{P_i} \cong P_i(A_S)$, $i = 1, \ldots, n$, the definition of natural projection and Lemma 2.6. The fourth equivalence is implied from the passivity of $\bar{E}_i$, $i = 1, \ldots, n$ and $EF1$-$EF4$, and finally, the last bisimilarity is due to the decomposability assumption of A_S. ∎

Remark 4.5 The significance of Theorem 4.2 is that under the passivity condition and $EF1$-$EF4$, although the local task automata may change after the failure (i.e., $F(P_i(A_S)) \ncong P_i(A_S)$), the team of agents can satisfy the global specification, as $\overset{n}{\underset{i=1}{||}} F(A_{P_i}||A_{C_i}) \cong \overset{n}{\underset{i=1}{||}} F(P_i(A_S)) \cong \overset{n}{\underset{i=1}{||}} P_i(A_S) \cong A_S$.

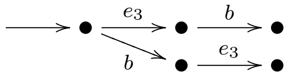
Example 4.8 This example illustrates a specification for a team of three agents that is globally satisfied and remains satisfied in spite of passive event failures, provided the conditions $EF1$-$EF4$. Consider a parallel distributed plant $A_P := \overset{3}{\underset{i=1}{||}} A_{P_i}$ with local plants A_{P_1}:



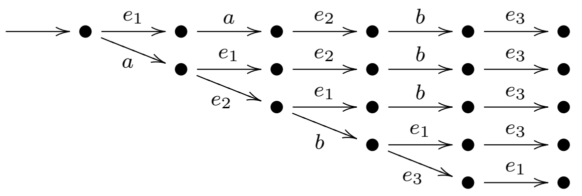
with $E_1 = \{a, e_1\}$, A_{P_2}:



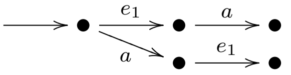
with $E_2 = \{a, b, e_2\}$, A_{P_3}:



with $E_3 = \{b, e_3\}$, having communication pattern $1 \in send_a(2)$, $3 \in send_b(2)$, and no more communication links. Assume that the global specification is given as A_S:

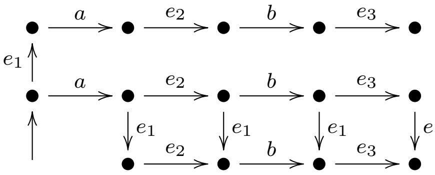


. A_S is decomposable, since the parallel composition of $P_1(A_S) \cong$



, $P_2(A_S) \cong$

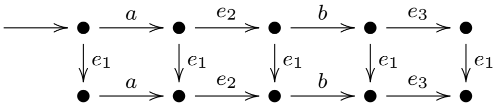
$\longrightarrow \bullet \xrightarrow{a} \bullet \xrightarrow{e_2} \bullet \xrightarrow{b} \bullet$ and $P_3(A_S) \cong$ $\longrightarrow \bullet \xrightarrow{b} \bullet \xrightarrow{e_3} \bullet$ is bisimilar to A_S. Now, taking the local controllers as $A_{C_i} := P_i(A_S)$, $i = 1, 2, 3$ results in $A_{P_i}||A_{C_i} \cong P_i(A_S)$, $i = 1, 2, 3$ and $\overset{3}{\underset{i=1}{||}} (A_{P_i}||A_{C_i}) \cong$



that is bisimilar to A_S, i.e., global specification is satisfied by designing local controllers A_{C_i} to satisfy local satisfactions $P_i(A_S)$.

Now, suppose that a fails in E_1. Since a is passive in E_1 and A_S satisfies $EF1$-$EF4$ (since $\delta(q_0, e_1ae_2be_3)! \wedge \delta(q_0, ae_1e_2be_3)!$ in A_S, and hence $EF1$ and $EF2$ are satisfied; $\Sigma_1 = \{e_1\}$, $\Sigma_2 = \{a, b, e_2\}$, $\Sigma_3 = \{b, e_3\}$ with the only shared events $b \in \Sigma_2 \cap \Sigma_3$, and the corresponding interleaving between $F(P_2(A_S)) \cong P_2(A_S)$ and $F(P_3(A_S)) \cong P_3(A_S)$ is ae_2be_3 that appears in A_S, with all permutations with e_1 from $F(P_1(A_S))$; hence, $EF3$ is satisfied, and finally, $EF4$ is fulfilled since $F(P_1(A_S))$, $F(P_2(A_S))$ and $F(P_3(A_S))$ are respectively bisimilar to automata $\longrightarrow \bullet \xrightarrow{e_1} \bullet$, $\longrightarrow \bullet \xrightarrow{a} \bullet \xrightarrow{e_2} \bullet \xrightarrow{b} \bullet$ and $\longrightarrow \bullet \xrightarrow{b} \bullet \xrightarrow{e_3} \bullet$ that all are deterministic. Therefore, according to Theorem 4.1, $\overset{3}{\underset{i=1}{||}} F(P_i(A_S)) \cong A_S$.

Moreover, since the failed event a is passive in E_1 and A_S satisfies $EF1$-$EF4$, as Theorem 4.2, the global specification remains satisfied after failure, as $\overset{3}{\underset{i=1}{||}} F(A_{P_i}||A_{C_i}) \cong \overset{3}{\underset{i=1}{||}} F(P_i(A_S)) \cong$



that is bisimilar to A_S.

4.4 Special Case: More Insight Into 2-Agent Case

This part provides a closer look into the two-agent case and illustrates the notion of global decision making after the event failures.

First, following lemma presents some properties on a 2-agent system that experiences passive failures. The properties will be then used to provide a deeper insight on the global decision making of the team on successive and adjacent transitions, in spite of passive failures.

Lemma 4.4 *Consider a deterministic task automaton $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$ and assume that A_S is decomposable with respect to parallel composition and natural projections P_i, $i = 1, 2$, and furthermore assume that $\bar{E}_i = \{a_{i,r}\}$, $r \in \{1, ..., n_i\}$ fail in E_i, $i \in \{1, 2\}$. If $\bar{E}_i$, $i \in \{1, 2\}$ are passive, then*

1. $\bar{E}_1 \cap \bar{E}_2 = \emptyset$;
2. $\bar{E}_1, \bar{E}_2 \subseteq E_1 \cap E_2$;
3. $\Sigma_1 \backslash \Sigma_2 = (E_1 \backslash E_2) \cup \bar{E}_2$ *and* $\Sigma_2 \backslash \Sigma_1 = (E_2 \backslash E_1) \cup \bar{E}_1$.

Proof: See the proof in the Appendix. ∎

Now, following lemma represents the conditions for maintaining the capability of a team of two cooperative agents for global decision making on the orders and selections of transitions in the global task automaton, after passive event failures.

Lemma 4.5 *Consider a deterministic task automaton $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$. Assume that A_S is decomposable, i.e., $A_S \cong P_1(A_S)||P_2(A_S)$, and furthermore, assume that $\bar{E}_i = \{a_{i,r}\}$ fail in E_i, $r \in \{1, ..., n_i\}$, and $\bar{E}_i$ are passive for $i \in \{1, 2\}$.*

Then, the following two expressions are equivalent:

- *(EF1 and EF2):* $\forall (e_1, e_2) \in \{(E_1 \backslash E_2, \bar{E}_1), (E_2 \backslash E_1, \bar{E}_2), (\bar{E}_1, \bar{E}_2)\}$, $q \in Q$, $s \in E^*$:

$$[\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!] \tag{4.1}$$

$$\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)! \tag{4.2}$$

- *($DC1_\Sigma$ and $DC2_\Sigma$):* $\forall e_1 \in \Sigma_1 \backslash \Sigma_2, e_2 \in \Sigma_2 \backslash \Sigma_1, q \in Q$, $s \in E^*$:

$$[\delta(q, e_1)! \wedge \delta(q, e_2)!] \Rightarrow [\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$$

$$\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!.$$

Proof: See the proof in the Appendix. ∎

Remark 4.6 $EF1$ and $EF2$ represent the decomposability conditions $DC1$ and $DC2$ after failure, i.e., for the refined local event sets Σ_1 and Σ_2. They say that after the failure, any decision on the switch or the order between two events that cannot be accomplished by at least one of the agents (neither $\{e_1, e_2\} \subseteq \Sigma_1$, nor $\{e_1, e_2\} \subseteq \Sigma_2$), then the decision should not be important (both orders should be legal). This is a good insight on the validity of $DC1$ and $DC2$ after the failure of passive events as it is stated in Lemma 4.5 and illustrated in Figure 4.2.

From Lemma 4.5, $(\Sigma_1 \backslash \Sigma_2) \times (\Sigma_2 \backslash \Sigma_1)$ is the union of four spaces: $(E_1 \backslash E_2) \times (E_2 \backslash E_1)$; $(E_1 \backslash E_2) \times (\bar{E}_1)$; $(\bar{E}_2) \times (E_2 \backslash E_1)$, and $(\bar{E}_1) \times (\bar{E}_2)$ (see Figure $4.2-(a)-(d)$). Note that due to Lemma 4.4, $\bar{E}_1 \cap \bar{E}_2 = \emptyset$.

Now, according to Theorem 2.1, for any pair of events from $(E_1 \backslash E_2) \times (E_2 \backslash E_1)$ (shown in Figure $4.2-(a)$), (4.1) and (4.2) are true as A_S is decomposable, before the failure. Moreover, (4.1) and (4.2) are also true for the pair of events from other three spaces of $(\Sigma_1 \backslash \Sigma_2) \times (\Sigma_2 \backslash \Sigma_1)$, due to $EF1$ and $EF2$ as it is elaborated as follows.

- Figure 4.2 – (b) shows $(E_1 \backslash E_2) \times (\bar{E}_1)$: any pair of events from this space contains in E_1, before the failure, but, contains in neither of E_1 and E_2 after the failure;
- Figure 4.2 – (c) depicts $(\bar{E}_2) \times (E_2 \backslash E_1)$: any pair of events from this space contains in E_2, before the failure, but, belongs to neither of E_1 and E_2 after the failure;
- Figure 4.2 – (d) illustrates $(\bar{E}_1) \times (\bar{E}_2)$: any pair of events from this space contains in both E_1 and E_2, before the failure, but, contains in none of them after the failure.

Therefore, since after the failure, for any pair of events from these three spaces, no agent can be responsible for decision making on switch/order between them (no local event set contains both events), then such decisions should not be important as it stated in $EF1$ and $EF2$.

Another implication of this result is that when the system is comprised of only two agents and one of those agent is failed, while all of its events are passive, then the task automaton remains decomposable as

Corollary 4.1 *Consider a deterministic task automaton $A_S = (Q, q_0, E = E_1 \cup E_2, \delta)$. Assume that A_S is decomposable, i.e., $A_S \cong P_1(A_S) || P_2(A_S)$. Assume furthermore that E_1 entirely fails, i.e., $\bar{E}_1 = E_1$. Then, $A_S \cong \overset{2}{\underset{i=1}{||}} F(P_i(A_S))$ if and only if $\bar{E}_1$ is passive.*

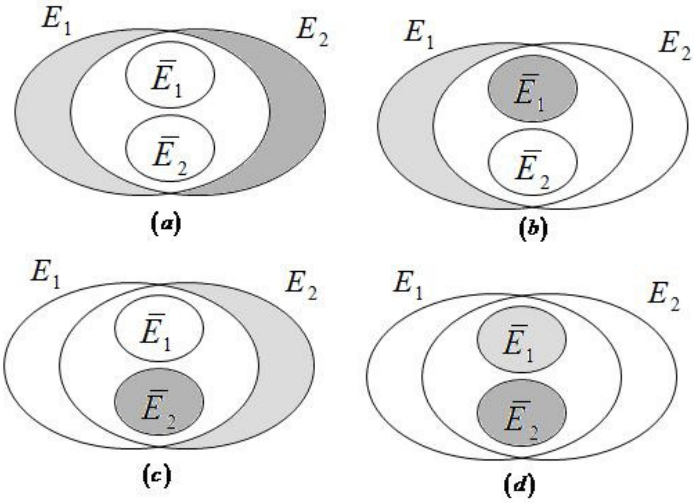


Figure 4.2: Illustration of $(\Sigma_1 \backslash \Sigma_2) \times (\Sigma_2 \backslash \Sigma_1)$.

Proof: **Sufficiency:** Since $\bar{E}_1 = E_1$, from the definition of passivity, Lemma 4.4 and $E = E_1 \cup E_2$, it follows that $E_1 \subseteq E_2 = E$ and $E_1 \backslash E_2 = \bar{E}_2 = \emptyset$; hence, $EF1$ and $EF2$ hold true, due to Lemma 4.5. Moreover, since $\Sigma_1 = E_1 \backslash \bar{E}_1 = \emptyset$, then $\Sigma_1 \backslash \Sigma_2 = \Sigma_1 = \emptyset$, that makes $EF3$ always true. Finally, by Lemma 4.1, $F(P_1(A_S))$ with $\Sigma_1 = \emptyset$ merges into its initial state, with no nondeterminism, and $F(P_2(A_S))$ with $\Sigma_2 = E$ is bisimilar to A_S which is deterministic, therefore, $EF4$ is satisfies, as well. This implies that when $\bar{E}_1 = E_1$, the passivity of $\bar{E}_1$ leads to $A_S \cong F(P_1(A_S))||F(P_2(A_S))$.

Necessity: The necessity is proven by contradiction. Suppose that $\bar{E}_1 = E_1$ and $A_S \cong F(P_1(A_S))||F(P_2(A_S))$, but $\exists e \in \bar{E}_1$, e is not passive in E_1. Then, from Lemma 4.1, it is follows that transitions on e cannot evolve in $F(P_1(A_S))||F(P_2(A_S))$, due to synchronization constraint in parallel composition; hence, $A_S \ncong F(P_1(A_S))||F(P_2(A_S))$ which is a contradiction. ■

4.5 Conclusion

This chapter proposed a method to investigate whether a decentralized bisimilarity control design remains valid, under the failure of some events in multi-agent systems. This work is a continuation of Chapters 2 and 3, in which necessary and sufficient conditions were given for task automaton decomposability; and the satisfaction of global specification was guaranteed upon the satisfaction of local specifications. This work defines a new notion of called passivity under which it is possible to transform the decentralized cooperative control problem under event failures into the standard decomposability problem in Chapters 2 and 3, and then identifies conditions to guarantee the supervised concurrent plant to still satisfy the global specification, in spite of event failures. The passivity of the failed events is turned to be a necessary condition for the task automaton to remain decomposable, and it is found to reflect the failure of redundant communication links. It is then proven that a decomposable task automaton remains decomposable, and hence satisfied, after some passive failures if and only if after the failures, the team of agents maintains the capability on collective decision making on the orders and selections of transitions and preserves the collective perceiving of the task such that the parallel composition of local task automata neither allows an illegal behavior (a string that is not in the global task automaton), nor disallows a legal behavior (a string from the global task automaton).

This result is of practical importance as it provides a sense of fault-tolerance to the task decomposability and top-down cooperative control of multi-agent systems, under event failures.

4.6 Appendix

4.6.1 Proof for Lemma 4.1

Firstly, in order to allow the global transitions, the failed event a in E_i has to be received from other agents not from its own sensors and actuator readings, otherwise, no local transitions on a evolve in either of $F(A_i)$ or $\overset{n}{\underset{i=1}{||}} F(A_i)$ (since other agents receive a from A_i). Therefore, the failed events have to necessarily be shared events ($loc(a) > 1$), and that after the failure of a in A_i, a is excluded from E_i, i.e., $\Sigma_i = E_i \backslash a$, as a is not received to A_i from other agents. Moreover, due to Definition 1.9, exclusion of a from E_i allows global transitions on a with no synchronization restriction from $F(A_i)$. Finally, the transitions on failed event a have to be replaced with ε-moves, in order to allow transitions after a in A_i, i.e., $\forall x_1, x_2 \in Q_i$, $x_2 \in \delta_i(x_1, a)$, then $[x_2]_{\Sigma_i} \in \delta_i^F([x_1]_{\Sigma_i}, a)$, $[x_1]_{\Sigma_i} = [x_2]_{\Sigma_i}$ and $F(A_i) = P_{E_i \backslash a}(A_i)$ (otherwise a transition of $\delta_i^F(\delta_i^F(x,a), e)$ will be disabled due to the stopping of execution of $\delta_i^F(x,a)$). It should be noted that, if there are no transitions after $\delta_i(x,a)$ (i.e., $\forall e \in E_i$: $\neg\delta_i(\delta_i(x,a), e)!$, then the stopping of $\delta_i(x,a)$ is identical to replacing this transition with an ε-move. These collectively mean that the preserving of global transitions in $\overset{n}{\underset{i=1}{||}} F(A_i)$ requires the local failures to be passive.

4.6.2 Proof for Lemma 4.2

Passivity of all $\bar{E}_i$, $i \in \{1, \ldots, n\}$, due to definition of passivity, leads to $\Sigma_i = E_i \backslash \bar{E}_i \subseteq E_i$; hence, the expression $[\exists E_i \in \{E_1, \cdots, E_n\}, \{e_1, e_2\} \subseteq E_i]$ in the antecedent of

$DC1$ and $DC2$ leads to $[\exists \Sigma_i \in \{\Sigma_1, \cdots, \Sigma_n\}, \{e_1, e_2\} \subseteq \Sigma_i]$, replacing E_i with $\Sigma_i = E_i \backslash \bar{E}_i$.

4.6.3 Proof for Lemma 4.3

Any nondeterminism in $F(P_i(A_S))$ appears either due to nondeterminism from $P_i(A_S)$ or newly formed nondeterminism because of replacing of passive events by ε.

In the first case, from the decomposability of A_S, $DC4$ says that for any $x, x_1, x_2 \in Q_i$, $e \in E_i \backslash \bar{E}_i$, $t \in E_i^*$, $x_1 \neq x_2$, $x_1 \in \delta_i(x, e)$, $x_2 \in \delta_i(x, e)$: $\delta_i(x_1, t)! \Leftrightarrow \delta_i(x_2, t)!$, i.e., $[x_1]_{\Sigma_i} \in \delta_i^F([x]_{\Sigma_i}, e)$, $[x_2]_{\Sigma_i} \in \delta_i^F([x]_{\Sigma_i}, e)$: $\delta_i^F([x_1]_{\Sigma_i}, p_{\Sigma_i}(t))! \Leftrightarrow \delta_i^F([x_2]_{\Sigma_i}, p_{\Sigma_i}(t))!$, which is $DC4$ for $F(P_i(A_S))$, with refined local event set Σ_i.

For the second case, any newly appeared nondeterminism is induced by transitions from the original local task automaton, in the following form. $\exists i \in \{1, \ldots, n\}, x, x_1, x_2 \in Q_i$, $t_1, t_2 \in \bar{E}_i^*$, $e \in E_i \backslash \bar{E}_i$, $t_1', t_2' \in E_i^*$, $x_1 \neq x_2$, $x_1 \in \delta_i(x, t_1 e)$, $x_2 \in \delta_i(x, t_2 e)$, t_1' transits after x_1 or x_2, but there does not exist t_2' after the other state (among x_1, x_2), such that $p_{E_i \backslash \bar{E}_i}(t_1') = p_{E_i \backslash \bar{E}_i}(t_2')$. In this case, $[x]_{\Sigma_i} = [\delta_i^F([x]_{\Sigma_i}, t_1)]_{\Sigma_i}$ $= [\delta_i^F([x]_{\Sigma_i}, t_2)]_{\Sigma_i}$; hence, $EF4$ becomes $[x_1]_{\Sigma_i} \in \delta_i^F([x]_{\Sigma_i}, e)$, $[x_2]_{\Sigma_i} \in \delta_i^F([x]_{\Sigma_i}, e)$: $\delta_i^F([x_1]_{\Sigma_i}, p_{\Sigma_i}(t_1'))! \Leftrightarrow \delta_i^F([x_2]_{\Sigma_i}, p_{\Sigma_i}(t_2'))!$, which is again equivalent to $DC4$ for $F(P_i(A_S))$.

4.6.4 Proof for Lemma 4.4

The first item is proven based on the fact that if $\exists e \in \bar{E}_1 \cap \bar{E}_2$, then $snd_e(1) = \emptyset \wedge snd_e(2) = \emptyset$ which is impossible, due to Remark 4.1 that requires $snd_e(i) =$

$\emptyset \wedge rec_e(i) \neq \emptyset$ for an event e to be passive in $E_i \in \{E_1, E_2\}$, in two-agent case.

The second item, comes from the passivity of $\bar{E}_1$ and $\bar{E}_2$ that implies that $\forall e \in \bar{E}_i$, $i = 1, 2$, $snd_e(i) = \emptyset \wedge rcv_e(i) \neq \emptyset$, and hence $loc(e) > 1$ which means $e \in E_j$, $j \in \{1, 2\} \backslash \{i\}$, i.e., $e \in E_1 \cap E_2$.

For the last item, from the second item and $\bar{E}_1 \cap \bar{E}_2 = \emptyset$ we respectively have $\bar{E}_1, \bar{E}_2 \subseteq E_1 \cap E_2$ and $\bar{E}_1 \subseteq \bar{E}_2'$, $\bar{E}_2 \subseteq \bar{E}_1'$ (In this proof, prime operation stands for the set complement, where the $E_1 \cup E_2$ is considered as the universal set). Consequently, $\Sigma_1 \backslash \Sigma_2 = (E_1 \backslash \bar{E}_1) \backslash (E_2 \backslash \bar{E}_2) = (E_1 \cap \bar{E}_1') \cap \big(E_2 \cap \bar{E}_2'\big)' = (E_1 \cap \bar{E}_1') \cap \big(E_2' \cup \bar{E}_2\big) = [(E_1 \cap \bar{E}_1') \cap E_2'] \cup [(E_1 \cap \bar{E}_1') \cap \bar{E}_2] = [E_1 \cap \big(\bar{E}_1 \cup E_2\big)'] \cup [(E_1 \cap \bar{E}_2) \cap \bar{E}_1'] = (E_1 \cap E_2') \cup (\bar{E}_2 \cap \bar{E}_1') = (E_1 \backslash E_2) \cup \bar{E}_2$. Similarly, $\Sigma_2 \backslash \Sigma_1 = (E_2 \backslash E_1) \cup \bar{E}_1$.

4.6.5 Proof for Lemma 4.5

To prove this lemma, firstly, we recall the decomposability result for two agents as stated in Theorem 2.1. In order to prove the equivalence of two cases in Lemma 4.5, one needs to prove that the set $\{\bar{E}_1 \times E_1 \backslash E_2, \bar{E}_2 \times E_2 \backslash E_1, \bar{E}_1 \times \bar{E}_2\}$ in $EF1$ and $EF2$ is equal to the set $\{(\Sigma_1 \backslash \Sigma_2) \times (\Sigma_2 \backslash \Sigma_1)\}$ in $DC1_\Sigma$ and $DC1_\Sigma$ (decomposability conditions $DC1$ and $DC2$ with respect to Σ_1 and Σ_2).

From Lemma 4.4, $e_1 \in \Sigma_1 \backslash \Sigma_2$, $e_2 \in \Sigma_2 \backslash \Sigma_1$ is equivalent to $e_1 \in (E_1 \backslash E_2) \cup \bar{E}_2$, $e_2 \in (E_2 \backslash E_1) \cup \bar{E}_1$ which means that $e_1 \in E_1 \backslash E_2 \vee e_1 \in \bar{E}_2$ and $e_2 \in E_2 \backslash E_1 \vee e_2 \in \bar{E}_1$, leading to four possible cases: $(e_1 \in E_1 \backslash E_2 \wedge e_2 \in E_2 \backslash E_1)$, $(e_1 \in E_1 \backslash E_2 \wedge e_2 \in \bar{E}_1)$, $(e_1 \in \bar{E}_2 \wedge e_2 \in E_2 \backslash E_1)$ or $(e_1 \in \bar{E}_2 \wedge e_2 \in \bar{E}_1)$.

Now, Lemma 4.5 is proven as follows. For the first case, since the decomposability of A_S implies $DC1$ and $DC2$, then, $\forall e_1 \in E_1 \backslash E_2, e_2 \in E_2 \backslash E_1$, $q \in Q$, $s \in E^*$: (4.1) and (4.2) hold true. For the second, third and fourth cases, i.e., when $(e_1 \in E_1 \backslash E_2 \wedge e_2 \in \bar{E}_1)$, $(e_1 \in \bar{E}_2 \wedge e_2 \in E_2 \backslash E_1)$ and $(e_1 \in \bar{E}_2 \wedge e_2 \in \bar{E}_1)$, then (4.1) and (4.2) are guaranteed by $EF1$ and $EF2$. Therefore, provided the decomposability of A_S, $EF1$ and $EF2$, (4.1) and (4.2) become true for all $e_1 \in \Sigma_1 \backslash \Sigma_2$, $e_2 \in \Sigma_2 \backslash \Sigma_1$. This means that $EF1$ and $EF2$ are respectively equivalent to $DC1$ and $DC2$ after failures (for Σ_1 and Σ_2).

Chapter 5

Event Distribution for Cooperative Tasking

5.1 Introduction

This chapter is a continuation of Chapters 2, 3 and 4, and aims to study whether one can modify the communication pattern between agents so as to make an originally indecomposable task become decomposable. It seems that a trivial solution to make any task automaton decomposable is to broadcast all private events and make them public. However, this is equivalent to the centralized control and impractical in many situations. An interesting follow-up question would be what is exactly needed to share by communication among agents such that an originally indecomposable task becomes decomposable. To answer this question, one needs to understand the causes of indecomposability and then find methods to overcome them.

For this purpose, we propose an algorithm that uses previous results on task decomposability in Chapters 2 and 3 to identify and overcome the dissatisfaction of each decomposability condition. The algorithm first removes all redundant communication links using the fault-tolerant result in Chapter 4. As a result, any violation of decomposability conditions, remained after this stage, is not due to redundant communication links, and hence cannot be removed by means of link deletions. Instead, the algorithm proceeds by establishing new communication links to provide enough information to facilitate the task automaton decomposition. Since each new communication link may overcome several violations of decomposability conditions, the algorithm may offer different options for link addition, leading to the question of optimal decomposability with the minimum number of communication links. It is found that if link additions impose no new violations of decomposability conditions, then it is possible to make the automaton decomposable with the minimum number of links. However, it is furthermore shown that, in general, new communication links may introduce new violations of decomposability conditions that in turn require establishing new communication links. In such cases, the optimal path depends on the structure of the automaton and requires a dynamic exhaustive search to find the sequence of link additions with the minimum number of links. Therefore, in case of new violations, a simple sufficient condition is proposed to provide a feasible suboptimal solution to enforce the decomposability, without checking of decomposability conditions after each link addition.

Similar problems on automaton decomposability have been also studied in computer science literature. For example, [69] characterized the conditions for decomposition of asynchronous automata in the sense of isomorphism based on the maximal cliques of the dependency graph. The work in [69] considers a set of events to be attributed to a number of agents, with no predefinition of local event sets. While event attribution is suitable for parallel computing and synthesis problems in computer science, control applications typically deal with parallel distributed plants [74] whose events are predefined by the set of sensors, actuators and communication links across the agents. Another related work is [72] that proposes a method for automaton decomposabilization by adding synchronization events such that the parallel composition of local automata is observably bisimilar to the original automaton. The method in [72], however, allows to add new synchronization events to the global event set that will enlarge the size of event set. Our work deals with those applications with fixed global event sets and a predefined distribution of events among local agents, where enforcing the decomposability is not allowed by adding new synchronization events, but instead by the redistribution of the existing events among the agents. This approach can decompose any deterministic task automaton, after which according to the previous results, the global specification is guaranteed to be satisfied, upon the satisfaction of local specifications.

The rest of the chapter is organized as follows. Problem formulation and a motivating examples are represented in Section 5.2. Section 5.3 proposes an algorithm to make any indecomposable deterministic automaton decomposable by modifying its local event sets. Illustrative examples are also given to elaborate the concept of task automaton decomposabilization. Finally, the chapter concludes with remarks and discussions in Section 5.4. Proofs of the lemmas are readily given in the Appendix.

5.2 Problem Formulation and Motivating Examples

In the previous chapters, we have identified the conditions for global task automaton decomposability and guaranteeing its satisfactions by the team as well as the conditions for the task to remain decomposable, in spite of event failures. In this chapter we are interested in the case that a task automaton is not decomposable and would like to ask whether it is possible to make it decomposable and if so, whether the automaton can be made decomposable with the minimum number of communication links. This problem is formally stated as

Problem 5.1 *(Event Distribution for Cooperative Tasking) Consider a deterministic task automaton A_S with event set $E = \overset{n}{\underset{i=1}{\cup}} E_i$ for n agents with local event sets E_i, $i = 1, \ldots, n$. If A_S is not decomposable, can we modify the sets of private and shared events between local event sets such that A_S becomes decomposable with respect to parallel composition and natural projections P_i, with the minimum number of communication links?*

One trivial way to make an automaton A decomposable, is to share all events among all agents, i.e., $E_i = E$, $\forall i = 1, \ldots, n$. This method , however, is equivalent to centralized control. In general, in distributed large scale systems, one of the objectives is to sustain the systems functionalities over as few number of communication links as possible, as will be addressed in the next section.

For more elaboration, let us to start with two motivating examples.

Example 5.1 Consider the plant of sequential belt conveyors and storage bin in Example 1.2. The task automaton is not decomposable with respect to parallel composition and natural projection P_i, $i \in \{A, B\}$, due to the violation of DC by successive private event pairs $\{B_{Start}, A_{Start}\}$ and $\{A_{Stop}, B_{Stop}\}$. To make A_S decomposable, $(B_{Start} \vee A_{Start}) \wedge (A_{Stop} \vee B_{Stop})$ should become common between E_A and E_B. Therefore, four options are possible: $(B_{Start} \wedge B_{Stop})$, $(B_{Start} \wedge A_{Stop})$, $(A_{Start} \wedge B_{Stop})$, or $(A_{Start} \wedge A_{Stop})$ become common. In each of these options two private events should become common; therefore, all four options are equivalent in the sense of optimality. Consider for example A_{Start} and A_{Stop} to become common. In this case the new local event sets are formed as $E_A = \{A_{Start}, Bin_{Full}, A_{Stop}\}$ and $E_B = \{B_{Start}, B_{Stop}, Bin_{Empty}, A_{Start}, A_{Stop}\}$. The automaton A_S will then become decomposable (i.e., $P_A(A_S)||P_B(A_S) \cong A_S$) with the new local event sets and corresponding local task automata as are shown in Figure 5.1.

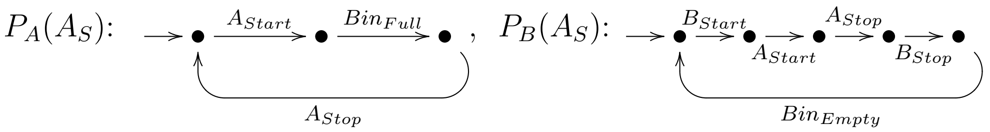


Figure 5.1: Local task automata for belt conveyors.

In this example, different sets of private events can be chosen to make A_S decomposable. All of these sets have the same cardinality; therefore, the optimal solution is not unique in this example. Next example shows a case with different choices of private event sets to be shared, suggesting optimal decomposition by choosing the set with the minimum cardinality.

Example 5.2 (Revisiting Examples 3.4 and 4.3 for task decomposabilization) Consider the task automaton in Section 3.3 and Examples 3.4 and 4.3, but with local event sets $E_1 = \{h_1, R_1toD_1, R_1onD_1, FWD, D_1opened, R_2in1, BWD, r\}$, $E_2 = \{h_2, R_2to2, R_2in2, R_2to1, R_2in1, r\}$, and $E_3 = \{h_3$, R_3to3, R_3in3, R_3toD_1, R_3onD_1, FWD, BWD, $D_1closed$, R_3to1, R_3in1, $r\}$. The task automaton is not decomposable with respect to the parallel composition and natural projections into these local event sets, due to the violation of $DC2$ as the decision on the order of event pairs $\{R_2in2, D_1opened\}$ and $\{D_1opened, R_2to1\}$ can be made by none of the agents.

To make it decomposable, one event among the set $\{R_2in2, D_1opened\}$ and another event among the set $\{D_1opened, R_2to1\}$ (either $\{D_1opened\}$ or $\{R_2in2, R_2to1\}$) should become common between E_1 and E_2. Therefore, in order for optimal decomposabilization, $\{D_1opened\}$ is chosen to become common due to its minimum cardinality. It is obvious that in this case only one event should become common while if $\{R_2in2, R_2to1\}$ was chosen, then two events were required to be shared. With the new event set $E_2 = \{h_2, R_2to2, R_2in2, D_1opened, R_2to1, R_2in1, r\}$, the automaton A_S becomes optimally decomposable, as it was shown in Example 4.3, with only one link addition.

Motivated by these examples, the core idea in our decompozabilization approach is to first check the decomposability of a given task automaton A_S, by Corollary 3.2, and if it is not decomposable, i.e., either of $DC1$-$DC4$ is violated then the proposed method is intended to make A_S decomposable, by eradicating the reasons of dissatisfying of decomposability conditions. We will show that the violation of decomposability conditions, can be rooted from two different sources: it can be because of over-communication among agents, that may lead to the violation of $DC3$ or/and $DC4$, or due to the lack of communication, that may lead to the violation of $DC1$, $DC2$, $DC3$ or/and $DC4$. Accordingly, decomposability can be enforced using two methods of link deletion and link addition, subjected to the type of indecomposability. Considering link deletion as an intentional event failure, according to Theorem 4.1 a link can be deleted only if it is passive and its deletion respects $EF1$-$EF4$. On the other hand, the second method of enforcing of decomposability, i.e., establishing new communication links, may result in new violations of $DC3$ or $DC4$, that should be treated, subsequently. Next part suggests basic rules to check and enforce each decomposability condition.

5.3 Task Automaton Decomposabilization

In order to proceed the approach, we firstly introduce four basic definitions to *detect* the components that contribute in the violation of each decomposability condition and then propose basic lemmas through which the communication links, and hence the local event sets are modified to *resolve* the violations of decomposability conditions.

5.3.1 Enforcing $DC1$ and $DC2$

This part deals with the enforcing of $DC1$ and $DC2$. For this purpose, the set of events that violate $DC1$ or $DC2$ is defined as follows.

Definition 5.1 ($DC1$&2-Violating set) Consider the global task automaton A_S with local event sets E_i for n agents such that $E = \bigcup_{i=1}^{n} E_i$. Then, the $DC1\&2$-Violating set operator $V : A_S \to E \times E$, indicates the set of unordered event pairs that violate $DC1$ or $DC2$ (violating pairs), and is defined as $V(A_S) := \{\{e_1, e_2\} | e_1, e_2 \in E, \forall E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \not\subset E_i, \exists q \in Q$ such that $\delta(q, e_1)! \wedge \delta(q, e_2)! \wedge \neg[\delta(q, e_1e_2)! \wedge \delta(q, e_2e_1)!]$ or $\neg[\delta(q, e_1e_2s)! \Leftrightarrow \delta(q, e_2e_1s)!]\}$, for some $s \in E^*$. Moreover, $W : A_S \to E$ is defined as $W(A_S) := \{e \in E | \exists e' \in E$ such that $\{e, e'\} \in V(A_S)\}$, and shows the set of events that contribute in $V(A_S)$ (violating events). For a particular event e and a specific local event set $E_i \in \{E_1, \ldots, E_n\}$, $W_e(A_S, E_i)$ is defined as $W_e(A_S, E_i) = \{e' \in E_i | \{e, e'\} \in V(A_S)\}$. This set captures the collection of events from E_i that pair up with e to contribute in the violation of $DC1$ or $DC2$. The cardinality of this set will serve as an index for the optimal addition of communication links to make $V(A_S)$ empty.

This definition suggests a way to remove a pair of events $\{e_1, e_2\}$ from $V(A_S)$, by sharing e_1 with one of the agents in $loc(e_2)$ or by sharing e_2 with one of the agents in $loc(e_1)$. Once there exist an agent that knows both event, $loc(e_1) \cap loc(e_2)$ becomes nonempty and e_1 and e_2 no longer contribute in the violation of $DC1$ or $DC2$ since $[\exists E_i \in \{E_1, \ldots, E_n\}, \{e_1, e_2\} \subseteq E_i]$ becomes true for e_1 and e_2 in Corollary 3.2. Therefore,

Lemma 5.1 *The set $V(A_S)$ becomes empty, if for any $\{e, e'\} \in V(A_S)$, e is included in E_i for some $i \in loc(e')$, or e' is included in E_j for some $j \in loc(e)$. In this case, $\{e, E_i\}$ or $\{e', E_j\}$ is called a $DC1\&2$-enforcing pair for $DC1\&2$-violating pair $\{e, e'\}$.*

Example 5.3 In Example 5.2, $V(A_S) = \{\{R_2in2, D_1opened\}, \{D_1opened, R_2to1\}\}$,

$W(A_S) = \{R_2in2, D_1opened, R_2to1\}$. Including $D_1opened$ in E_2 vanishes $V(A_S)$ and makes A_S decomposable.

Therefore, as it is shown in Example 5.2, applying Lemma 5.1 may offer different options for event sharing, since pairs in $V(A_S)$ may share some events. In this case, the minimum number of event conversions would be obtained by forming a set of events that are most frequently shared between the violating pairs. This gives the minimum cardinality for the set of private events to be shared, leading to the minimum number of added communication links. Such a choice of events offers a set of events that span all violating pairs. These pairs are captured by $W_e(A_S, E_i)$ for any event e. In order to minimize the number of added communication links for vanishing $V(A_S)$, one needs to maximize the number of deletions of pairs from $V(A_S)$ per any link addition. For this purpose, for any event e, $W_e(A_S, E_i)$ is formed to understand the frequency of appearance of e in $V(A_S)$ for any E_i, and then, the event set E_i with maximum $|W_e(A_S, E_i)|$ is chosen to include e (Here, $|.|$ denotes the set's cardinality). In this case, the inclusion of e in E_i will delete as many pairs as possible from $V(A_S)$.

Interestingly, these operators can be represented using graph theory as follows. A graph $G = (W, \Sigma)$ consists of a node (vertex) set W and an edge set Σ, where an edge is an unordered pair of distinct vertices. Two nodes are said to be adjacent if they are connected through an edge, and an edge is said to be incident to a node if they are connected. The valency of a node is then defined as the number of its incident edges [139]. Now, since we are interested in removing the violating pairs by making one of their events to be shared, it is possible to consider the violating events as the nodes of a graph such that two nodes are adjacent in this graph when they form a violating pair. This graph is formally defined as follows.

Definition 5.2 ($DC1\&2$-Violating graph) Consider a deterministic automaton A_S. The $DC1\&2$-Violating graph, corresponding to $V(A_S)$, is a graph $G(A_S) = (W(A_S), \Sigma)$. Two nodes e_1 and e_2 are adjacent in this graph when $\{e_1, e_2\} \in V(A_S)$.

In this formulation, the valency of each node e with respect to a local event set $E_i \in \{E_1, \dots, E_n\}$ is determined by $val(e, E_i) = |W_e(A_S, E_i)|$. When e is included into E_i, it means that all violating pairs containing e and events from E_i are removed from $V(A_S)$, and equivalently, all corresponding incident edges are removed from $G(A_S)$. For this purpose, following algorithm finds the set with the minimum number of private events to be shared, in order to satisfy $DC1$ and $DC2$. The algorithm is accomplished on graph $G(A_S)$, by finding e and E_i with maximum $|W_e(A_S, E_i)|$ and including e in E_i, deleting all edges from e to E_i, updating $W(A_S)$, and continuing until there is no more edges in $G(A_S)$ to be deleted.

Algorithm 5.1

1. *For a deterministic automaton A_S, with local event sets E_i, $i = 1, \dots, n$, violating DC1 or DC2, form the DC$1\&2$-Violating graph ; set $E_i^0 = E_i$, $i = 1, \dots, n$; $V^0(A_S) = V(A_S)$; $W^0(A_S) = W(A_S)$; $G^0(A_S) = (W(A_S), \Sigma)$; k=1;*
2. *among all events (nodes) in $W^{k-1}(A_S)$, find e with the maximum $|W_e^{k-1}(A_S, E_i^{k-1})|$, for all $E_i^{k-1} \in \{E_1^{k-1}, \dots, E_n^{k-1}\}$;*
3. *$E_i^k = E_i^{k-1} \cup \{e\}$; and delete all edges from e to E_i^k;*

4. *update $W_e^k(A_S, E_i)$ for all nodes of $G(A_S)$;*

5. *set $k = k + 1$ and go to step (2);*

6. *continue, until there exist no edges.*

Remark 5.1 The complexity of the algorithm is of the order of $O(|E|^4 \times |Q|^3)$, where $|E|$ and $|Q|$ respectively denote the sizes of the state space and global event set. The algorithm successfully terminates due to the finite set of edges and nodes in the graph $G(A_S)$ and enforces A_S to satisfy $DC1$ and $DC2$ as the following lemma.

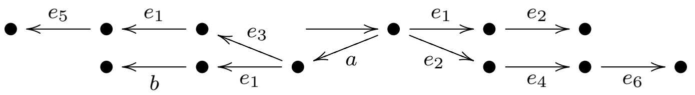
Lemma 5.2 *Algorithm 5.1 leads A_S to satisfy $DC1$ and $DC2$ with the minimum addition of communication links. Moreover if A_S satisfies $DC3$ and $DC4$ and $E_i^k = E_i^{k-1} \cup \{e\}$ in Step 3 does not violate $DC3$ and $DC4$ in all iterations, then Algorithm 5.1 makes A_S decomposable with the minimum addition of communication links.*

Proof: See the Appendix for proof. ∎

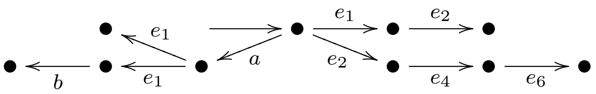
Remark 5.2 (Special Case: Two Agents) For the case of two agents, since there are only two local event sets, for all $\{e, e'\} \in V(A_S)$, e and e' are from different local event sets; therefore, for $n = 2$, $|W_e(A_S, E_i)|$ is equivalent to $val(e)$, and the addition of e into E_i in each step implies the deletion of all incident edges of e.

Remark 5.3 Although Algorithm 5.1 leads A_S to satisfy $DC1$ and $DC2$, it may cause new violations of $DC3$ or/and $DC4$, due to establishing new communication links.

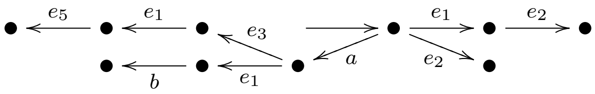
Example 5.4 Consider a task automaton A_S:



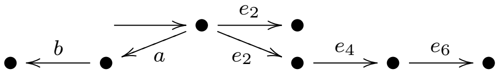
with local event sets $E_1 = \{a, b, e_1, e_3, e_5\}$ and $E_2 = \{a, b, e_2, e_4, e_6\}$. Both $DC1$ and $DC2$ are violated by event pair $\{e_1, e_2\}$ when they require a decision on a choice and a decision on their order from the initial state, while none of the agents knows both of them. To vanish $V(A_S) = \{\{e_1, e_2\}\}$, two enforcing pairs are suggested: $\{e_1, E_2\}$ (e_1 to be included in E_2) or $\{e_2, E_1\}$ (e_2 to be included in E_1). However, the inclusion of e_1 in E_2, cause a new violation of $DC4$ since with new $E_2 = \{a, b, e_1, e_2, e_4, e_6\}$, $P_2(A_S)$ is obtained as $P_2(A_S)$:



, violating $DC4$, due to new nondeterminism, for which e_3 also is required to be included to E_2 in order to make A_S decomposable. On the other hand, if instead of including e_1 in E_2, one included e_2 in E_1, then besides a violation of $DC4$ (as there does not exists a deterministic automaton that bisimulates $P_2(A_S)$), a violation of $DC3$ emerges, as with new event set $E_1 = \{a, b, e_1, e_2, e_3, e_5\}$, the parallel composition of $P_1(A_S)$:



and $P_2(A_S)$:



produces string $e_1e_2e_4e_6$ that does not appear in A_S. To make A_S decomposable, we also need to include e_1 and e_3 in E_2.

5.3.2 Enforcing $DC3$

Lemma 5.1 proposes adding communication links to make $DC1$ and $DC2$ satisfied. Next step is to deal with the violations of $DC3$. In contrast to the cases for $DC1$ and $DC2$, the violation of $DC3$ can be overcome either by disconnecting one of its communication links to prevent the illegal synchronization of strings, or by introducing new shared events to fix strings and avoid illegal interleavings.

To handle the violations of $DC3$, we firstly define the set of transitions that violate $DC3$ as follows.

Definition 5.3 ($DC3 - \textit{violating}$ Tuples) Consider a deterministic automaton A_S, satisfying $DC1$ and $DC2$ and let $\tilde{L}\left(A_S\right) \subseteq L\left(A_S\right)$ be the largest subset of $L\left(A_S\right)$ such that $\forall s \in \tilde{L}\left(A_S\right), \exists s' \in \tilde{L}\left(A_S\right), \exists E_i, E_j \in \left\{E_1, ..., E_n\right\}, i \neq j, p_{E_i \cap E_j}\left(s\right)$ and $p_{E_i \cap E_j}\left(s'\right)$ start with the same event $a \in E_i \cap E_j$. For any such E_i, E_j and a, if $\exists \{s_1, \cdots, s_n\} \in L\left(A_S\right)$, $\exists s_i, s_j \in \{s_1, \cdots, s_n\}, s_i \neq s_j$, $s_i, s_j \in \tilde{L}\left(A_S\right)$, $\neg\delta(q_0, \overset{n}{\underset{i=1}{|}} \overline{p_i\left(s_i\right)})!$, then a is called a $DC3 - \textit{violating}$ event with respect to s_1, s_2, E_i and E_j, and (s_1, s_2, a, E_i, E_j) is called a $DC3$-violating tuple. The set of all $DC3 - \textit{violating}$ tuples is denoted by $DC3 - V$ and defined as $DC3 - V = \{(s_1, s_2, a, E_i, E_j) | e$ is a DC3-violating event with respect to s_1, s_2, E_i and E_j $\}$.

Any violation in $DC3$ can be interpreted in two ways: firstly, it can be seen as an over-communication of shared event a that leads to the synchronization of s_1 and s_2 in (s_1, s_2, a, E_i, E_j) and emerging illegal interleaving strings from the composition of $P_i(A_S)$ and $P_j(A_S)$. In this case, if event a is excluded from E_i or E_j, then a will no longer contribute in synchronization to generate illegal interleavings; therefore, (s_1, s_2, a, E_i, E_j) will no longer remain a $DC3$-violating tuple. However, the exclusion of a from E_i or E_j is allowed, only if it is passive (exclusion is considered as an intentional event failure) and does not violate $EF1$-$EF4$. The second interpretation reflects a violation of $DC3$ as a lack of communication, such that if for any $DC3$ violating tuple (s_1, s_2, a, E_i, E_j), one event that appears before a in s_1 or s_2, is shared between E_j and E_j, then $P_i(A_S)$ and $P_j(A_S)$ will have enough information to distinguish s_1 and s_2 to prevent the illegal interleaving of strings. Two methods for resolving the violation of $DC3$ can be therefore stated as the following lemma.

Lemma 5.3 *Consider an automaton A_S, satisfying $DC1$ and $DC2$. Then any $DC3$-violating tuple (s_1, s_2, a, E_i, E_j) is overcome, when:*

1. *a is excluded from E_i or E_j (eligible if it respects passivity and $EF1$-$EF4$), or*
2. *if $\exists b \in (E_i \cup E_j)\backslash(E_i \cap E_j)$ that appears before a in only one of s_1 and s_2, then b is included in $E_i \cap E_j$, otherwise, two events $e_1 \in p_{E_i \cup E_j}(s_1)$, $e_2 \in p_{E_i \cup E_j}(s_2)$, such that $e_1 \neq e_2$, e_1, e_2 appear before a in s_1 and s_2, are included in $E_i \cap E_j$.*

Proof: See the proof in the Appendix. ∎

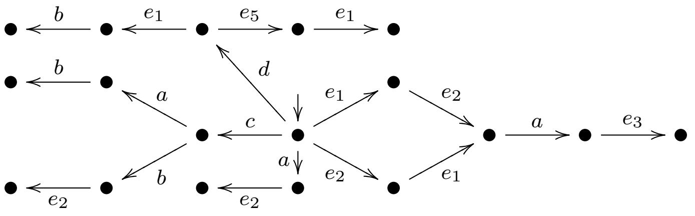
To handle a violation of $DC3$, when, $b \in E_i \backslash E_j$ is to be included in E_j, then $\{b, E_j\}$ is called a $DC3$-enforcing pair; while, when $\{e_1, e_2\} \subseteq E_i \backslash E_j$ has to be included in E_j, then $\{\{e_1, e_2\}, E_j\}$ is denoted as a $DC3$-enforcing tuple. Finally, when $e_1 \in E_i \backslash E_j$ and $e_2 \in E_j \backslash E_i$ have to be included in E_j and E_i, respectively, then $\{\{e_1, E_j\}, \{e_2, E_i\}\}$ is called a $DC3$-enforcing tuple.

Remark 5.4 Applying the first method in Lemma 5.3, namely, the exclusion of a from E_i or E_j in a $DC3$-violating tuple (s_1, s_2, a, E_i, E_j), is only allowed if a is passive in that local event set, and the exclusion does not violate $EF1$-$EF4$. The reason is that once a shared event $a \in E_i \cap E_j$ becomes a private one in for example E_i, then decision makings on the order/selection between any $e \in E_i \backslash a$ and a cannot be accomplished by the $i - th$ agent, and if there is no other agent to do so, then A_S

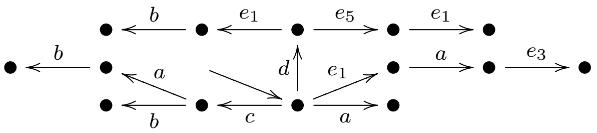
becomes indecomposable. Moreover, the deletion of a communication link may also result in the generation of new interleavings in the composition of local automata, that are not legal in A_S (violation of $EF3$). In addition, the deletion of a from E_i may impose a nondeterminism in bisimulation quotient of $P_i(A_S)$, leading to the violation of $EF4$. On the other hand, the second method, namely, establishing new communication link by sharing b with E_i or E_j may lead to new violations of $DC3$ or $DC4$ that have to be avoided or resolved, subsequently.

Both methods in Lemmas 5.3 present ways to resolve the violation of $DC3$. They differ however in the number of added communication links, as the first method deletes links, whereas the second approach adds communication links to enforce $DC3$. Therefore, in order to have as few number of links as possible among the agents, one should start with the link deletion method first, and if it is not successful due to the violation of passivity or any of $EF1$-$EF4$, then link addition is used to remove $DC3$-violating tuples from $DC3 - V$.

Example 5.5 This example shows an indecomposable automaton that suffers from a conflict on a communication link whose existence violates $DC3$, whereas its deletion dissatisfies $EF1$, $EF2$ and $EF4$. Consider the task automaton A_S: with communication pattern $2 \in snd_{\{a,b,c,d\}}(1)$, $1 \notin snd_{\{a,b,c,d\}}(2)$ and local event sets $E_1 = \{a, b, c, d, e_1, e_3, e_5\}$,

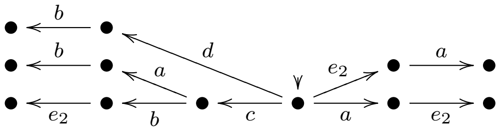


$E_2 = \{a, b, c, d, e_2\}$, leading to $P_1(A_S)$:

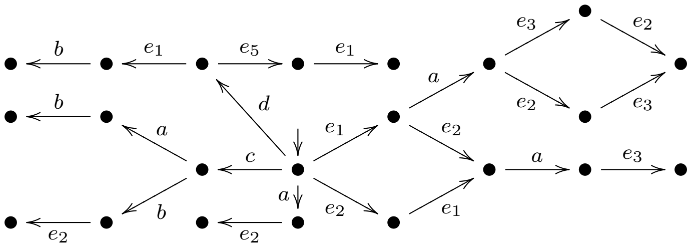


,

$P_2(A_S)$:

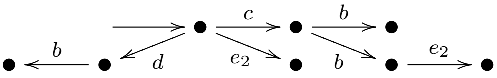


and $P_1(A_S)||P_2(A_S)$:



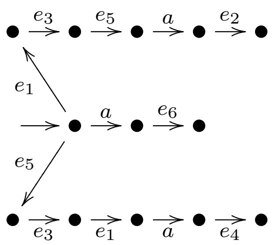
which is not bisimilar to A_S.

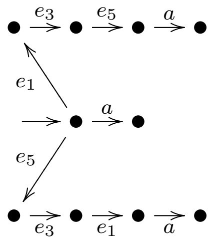
Here, A_S is not decomposable since two strings $e_1ae_2e_3$ and $e_1ae_3e_2$ are newly generated from the interleaving of strings in $P_1(A_S)$ and $P_2(A_S)$, while they do not appear in A_S; hence, $DC3$ is not fulfilled, due to $DC3$-violating tuples $(e_1e_2ae_3, ae_2, a, E_1, E_2)$ and $(e_2e_1ae_3, ae_2, a, E_1, E_2)$. Now, as Lemma 5.3, one way to fix the violation of $DC3$ is by excluding a from E_2. However, although a is passive in E_2, its exclusion from E_2 dissatisfies $EF1$(as $\delta(q_0, e_2)! \wedge \delta(q_0, a)! \wedge \neg[\delta(q_0, e_2a)! \wedge \delta(q_0, ae_2)!]$) and $EF2$ (since $\delta(q_0, e_1e_2a)! \wedge \neg\delta(q_0, e_1ae_2)!$) and also $\delta(q_0, ae_2)! \wedge \neg\delta(q_0, e_2a)!$. In this case, $DC4$ also will be violated as $P_2(A_S)$ becomes $P_2(A_S) \cong$

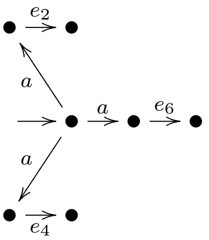


that bisimulates no deterministic automaton.

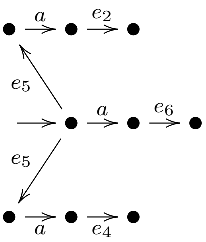
Lemma 5.3 also suggests another method to enforce $DC3$, by including either e_1 in E_2 or e_2 in E_1. The inclusion of e_1 in E_2, however, leads to another violation of $DC4$, as it produces a nondeterminism after event d. This in turn will need to include e_5 in E_2 to make A_S decomposable. Alternatively, instead of inclusion of e_1 in E_2, one can include e_2 in E_1, that enforces $DC3$ and makes A_S decomposable. The second method of Lemma 5.3 is more elaborated in the next example.

Example 5.6 This example shows the handling of $DC3$-violating tuples using the second method in Lemma 5.3, i.e., by event sharing. Later on, this example will be also used to illustrate the enforcement of $DC4$. Consider a task automaton A_S: 

with local event sets $E_1 = \{a, e_1, e_3, e_5\}$ and $E_2 = \{a, e_2, e_4, e_6\}$, and let three branches in A_S from top to bottom to be denoted as $s_1 := e_1e_3e_5ae_2$, $s_3 := ae_6$ and $s_2 := e_5e_3e_1ae_4$. This automaton does not satisfy $DC4$ (as $P_2(A_S)$ has no deterministic bisimilar automaton), as well as $DC3$, as the parallel composition of $P_1(A_S)$: 

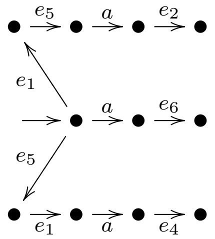
and $P_2(A_S)$: 

have illegal interleaving strings $\{e_1e_3e_5ae_4, e_5e_3e_1ae_2\}$, $e_1e_3e_5ae_6$ and $e_5e_3e_1ae_6$, corresponding to $DC3$-violating tuples (s_1, s_2, a, E_1, E_2), (s_1, s_3, a, E_1, E_2) and $(s_2, s_3,\ a, E_1, E_2)$, respectively.

For pairs of strings $\{s_1, s_3\}$ and $\{s_2, s_3\}$, there exits an event $e_5 \in (E_1 \cup E_2)\backslash(E_1 \cap E_2)$ that appears before a, only in s_1 and s_2, but not in s_3. Therefore, the inclusion of e_5 in E_2, removes the illegal interleavings between s_1 and s_2 with s_3, but not across s_1 and s_2, as with new $E_2 = \{a, e_2, e_4, e_5, e_6\}$ and $P_2(A_S)$: 

, (s_1, s_3, a, E_1, E_2) and (s_2, s_3, a, E_1, E_2) are no longer $DC3$-violating tuples, while (s_1, s_2, a, E_1, E_2) still remains a $DC3$-violating one with illegal interleavings $e_1e_3e_5ae_4$

and $e_5e_3e_1ae_2$. The reason is that e_5 appears before a in both s_1 and s_2, and there is no event that appear before a only in one of the strings s_1 and s_2. For this case, according to Lemma 5.3, two different events that appear before "a", one from $p_{E_1 \cup E_2}(s_1) = s_1$ and the other from $p_{E_1 \cup E_2}(s_2) = s_2$, say e_1 and e_5 have to be attached to E_2, resulting in $E_2 = \{a, e_1, e_2, e_4, e_5, e_6\}$, and $P_1(A_S)||P_2(A_S) \cong A_S$.



5.3.3 Enforcing $DC4$

Similar to $DC1$-$DC3$, a violation of $DC4$ can be regarded as a lack of communication link that causes nondeterminism in a local task automaton. Such interpretation calls for establishing a new communication link to prevent the emergence of local nondeterminism. Moreover, when this local nondeterminism occurs on a shared event, the corresponding violation of $DC4$ can be overcome by excluding the shared event from the respective local event set. It should be noted however that the event exclusion should respect the passivity and $EF1$-$EF4$ conditions. Moreover, when $DC4$ is enforced by link additions, similar to what we discussed for $DC3$, the addition of new communication link may cause new violations of $DC3$ or/and $DC4$. To enforce $DC4$, firstly a $DC4$-violating tuple is defined as follows.

Definition 5.4 ($DC4 - violating$ Tuple) Consider a deterministic automaton A_S with local event sets $E_i = 1, \ldots, n$, $\forall i \in \{1, ..., n\}$, $q, q_1, q_2 \in Q$, $t_1, t_2 \in (E \backslash E_i)^*$, $e \in E_i$, $\delta(q, t_1e) = q_1 \neq \delta(q, t_2e) = q_2$, $\exists t \in E^*$, $\delta(q_1, t)!$, but $\nexists t' \in E^*$ such that

$\delta(q_2, t')!$, $p_i(t) = p_i(t')$. Then, (q, t_1, t_2, e, E_i) is called a $DC4$-violating tuple.

This definition suggests a way to overcome the violation of $DC4$, as stated in the following lemma.

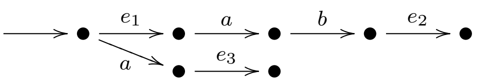
Lemma 5.4 *Any $DC4$-violating tuple (q, t_1, t_2, e, E_i) is overcome, when:*

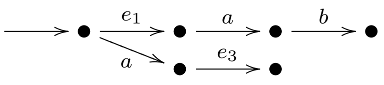
1. *e is excluded from E_i, (eligible, if it is passive in E_i and its exclusion respects $EF1 - EF4$), or*
2. *if $\exists e' \in (t_1 \cup t_2)\backslash(t_1 \cap t_2)$, e' is included in E_i; otherwise, $e_1 \in t_1$ and $e_2 \in t_2$, such that $e_1 \neq e_2$, are included in E_i. In these cases, $\{e', E_i\}$ and $\{\{e_1, e_2\}, E_i\}$ are called $DC4$-enforcing tuples.*

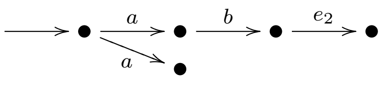
Proof: See the proof in the Appendix. ∎

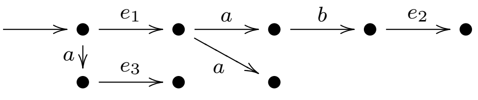
Following examples illustrate the methods in Lemma 5.4 to enforce $DC4$.

Example 5.7 This example shows an automaton that is indecomposable due to a violation in $DC4$, while $DC4$ can be enforced using both methods: event exclusion as well as event inclusion.

Consider the task automaton A_S: 

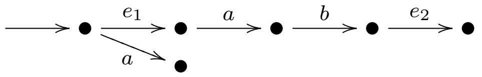
with $E_1 = \{a, b, e_1, e_3\}$, $E_2 = \{a, b, e_2\}$, $2 \in snd_{\{a,b\}}(1)$, $1 \notin snd_{\{a,b\}}(2)$, leading to $P_1(A_S)$: 

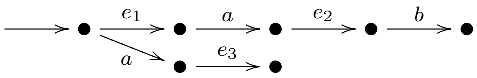
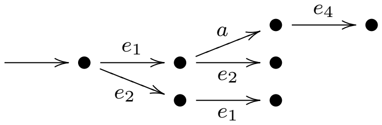
, $P_2(A_S)$: 

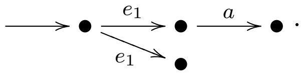
, and $\overset{2}{\underset{i=1}{||}} P_i(A_S) \cong$ 

which is not bisimilar to A_S, due to the violation of $DC4$ as there does not exist a deterministic automaton $P_2'(A_S)$ such that $P_2'(A_S) \cong P_2(A_S)$. Here, $(q_0, t_1 = e_1, t_2 = \varepsilon, a, E_2)$ is a $DC4$-violating tuple.

Since a is passive in E_2 and its exclusion from E_2 keeps $EF1$-$EF4$ valid, according to Lemma 5.4, one way to enforce $DC4$ is the exclusion of a from E_2, resulting in $E_2 = \{b, e_2\}$, $P_2(A_S)$: $\longrightarrow\bullet\xrightarrow{b}\bullet\xrightarrow{e_2}\bullet$ and $P_1(A_S)||P_2(A_S) \cong A_S$.

Another suggestion of Lemma 5.4 to overcome the $DC4$-violating tuple $(q_0, t_1 = e_1, t_2 = \varepsilon, a, E_2)$ is the addition of a communication link to prevent the nondeterminism in $P_2(A_S)$. Since there exists e_1 that appears before a in t_1 only, the inclusion of e_1 in E_2 also enforces $DC4$ as with new $E_2 = \{a, b, e_1, e_2\}$, $P_2(A_S)$:  and $\overset{2}{\underset{i=1}{||}} P_i(A_S) \cong A_S$. For the cases that there does not exist an event that appears before a in only one of the strings t_1 or t_2, according to Lemma 5.4, one needs to attach one event from each of two strings t_1 and t_2 in E_i. For instance consider the $DC4$-violating tuple $(t_1 = e_1e_3e_5, t_2 = e_5e_3e_1, a, E_2)$ in Example 5.6, with no event that appears before a in $(t_1 \cup t_2)\backslash(t_1 \cap t_2)$. In that case $\{e_1 \in t_1, e_5 \in t_2\}$ could be included in E_2 to make A_S decomposable, as it was shown in Example 5.6.

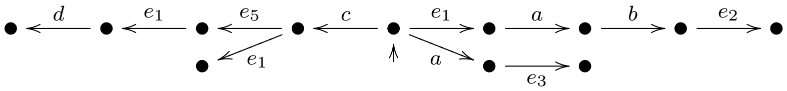
Example 5.8 This example shows a violation of $DC4$ that can be overcome by link addition, but not using link deletion method. Example 5.7 showed a violation of $DC4$ that could be handled using both methods in Lemma 5.4, namely, by link addition and link deletion. In Example 5.7, event a was a passive shared event whose exclusion from E_2 respected $EF1$-$EF4$, otherwise it was not allowed to be excluded. If the task automaton was  with $E_1 = \{a, b, e_1, e_3\}$, $E_2 = \{a, b, e_2\}$, then $DC4$ could not be enforced by the exclusion of a from E_2, as $EF2$ was violated since after this exclusion, no agent can handle the decision making on the order of a and e_2. Another constraint for link deletion is the passivity of the event. For example, consider A'_S: 

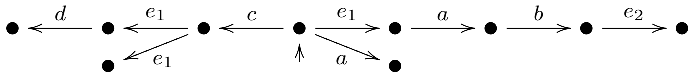
with $E_1 = \{e_1, a\}$, $E_2 = \{e_2, e_4, a\}$. A'_S is not decomposable due to the violation of $DC4$ in $P_1(A'_S)$: 

. The nondeterminism in $P_1(A_S)$, and accordingly the $DC4$-violating tuple $(q_0, \varepsilon, e_2, e_1, E_1)$, cannot be removed by event exclusion since it occurs on e_1 that is not a shared event. To enforce $DC4$, according to Lemma 5.4, e_2 is required to be included into E_1 that makes A'_S decomposable.

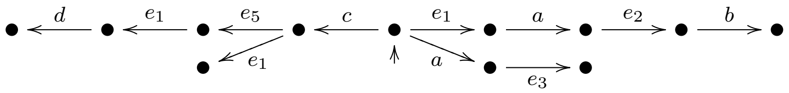
Another important issue for the addition of communication link to enforce $DC4$ is that establishing new communication link may lead to new violations of $DC3$ or $DC4$, as it is shown in the following example.

Example 5.9 Assume the task automaton in Example 5.7 had a part as shown in the left hand side of the initial state in A_S:



with $E_1 = \{a, b, c, d, e_1, e_3, e_5\}$, $E_2 = \{a, b, c, d, e_2\}$. Identical to Example 5.7, $(q_0, t_1 = e_1, t_2 = \varepsilon, a, E_2)$ is a $DC4$-violating tuple and can be overcome by excluding a from E_2, removing the nondeterminism on a in $P_2(A_S)$. However, unlike Example 5.7, including e_1 into E_2 (i.e., $E_2 = \{a, b, c, d, e_1, e_2\}$), leads to a new violation of $DC4$ in $P_2(A_S)$: 

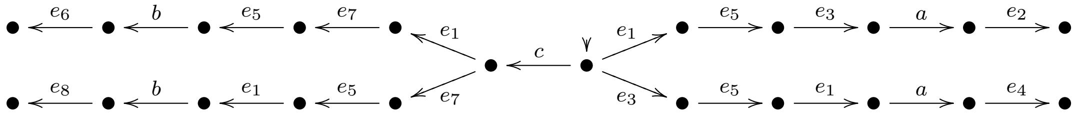
, with a $DC4$-violating tuple $(\delta(q_0, c)$, e_5, ε, e_1, $E_2)$, that in turn requires the attachment of e_5 to E_2, in order to enforce $DC4$.

If in this example, the order of e_2 and b was reverse, i.e., the task automaton was A'_S: 

with $E_1 = \{a, b, c, d, e_1, e_3,\ e_5\}$, $E_2 = \{a, b, c, d, e_2\}$. Then as it was shown in Example 5.8, the $DC4$-violating tuple $(q_0, e_1, \varepsilon,\ a, E_2)$ could not be dealt with the exclusion of a from E_2, due to $EF2$, neither by the inclusion of e_1 into E_2 (since as it was mentioned for A_S in this example, it generates a new violation of $DC4$ that consequently requires another inclusion of e_5 into E_2 to satisfy $DC4$).

Remark 5.5 Both Lemmas 5.3 and 5.4 provide sufficient conditions for resolving the violations of $DC3$ and $DC4$, respectively. They do not however provide the necessary solutions, neither the optimal solutions, as illustrated in the following example. We will show that for $DC3$ and $DC4$, in general one requires to search exhaustively to find the optimal sequence of enforcing tuples, to have the minimum number of link additions. In this case, instead of exhaustive search for optimal solution, it is reasonable to introduce sufficient conditions to provide a tractable procedure for a feasible solution to make an automaton decomposable.

Example 5.10 Consider a task automaton A_S:

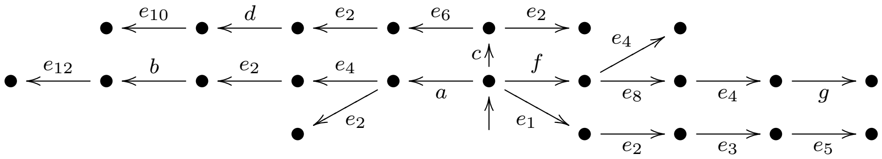


with local event sets $E_1 = \{a, b, c, e_1, e_3, e_5, e_7\}$ and $E_2 = \{a, b, c, e_2, e_4, e_6, e_8\}$. A_S is indecomposable due to $DC3$-violating tuples $(e_1e_5e_3ae_2, e_3e_5e_1ae_4, a, E_1,\ E_2)$ and $(e_1e_7e_5be_6,\ e_7e_5e_1be_8,\ a,\ E_1,\ E_2)$ and $DC4$-violating tuples $(q_0,\ e_1e_5e_3,\ e_3e_5e_1,\ a,\ E_2)$ and $(\delta(q_0, c), e_1e_7e_5, e_7e_5e_1, b, E_2)$. According to Lemmas 5.3 and 5.4, two enforcing tuples $\{\{e_1, e_3\}, E_2\}$ and $\{\{e_1, e_7\}, E_2\}$ remove all violations of $DC3$ and $DC4$. However, this solution is not unique, nor optimal, as the enforcing tuple $\{\{e_1, e_5\}, E_2\}$ enforced $DC3$ and $DC4$ with the minimum number of added communication links.

5.3.4 Exhaustive Search for Optimal Decompozabilization

Another difficulty is that enforcing the decomposability conditions using link deletion is limited to passivity and $EF1$-$EF4$, and after the deletions of redundant links (that are passive and their deletion respect $EF1$-$EF4$), the only way to make the automaton decomposable is to establish new communication links. Addition of new links, on the other hand, may lead to new violations of $DC3$ or $DC4$ (as illustrated in Examples 5.5 and 5.9), whose resolution in turn may introduce new violations. It means that, in general, the resolution of decomposability conditions can dynamically result in new violations of decomposability conditions, as it is elaborated in the following example.

Example 5.11 Consider the task automaton A_S:



with local event sets $E_1 = \{a, b, c, d, f, g, e_1, e_3, e_5\}$ and $E_2 = \{a, b, c, d, f, g, e_2, e_4, e_6, e_8, e_{10}, e_{12}\}$. This automaton is indecomposable due to $DC2$-violating event pairs $\{(e_1, e_2), (e_2, e_3)\}$ with the corresponding enforcing tuple $\{e_1, E_2\}$, $\{e_3, E_2\}$ and $\{e_2, E_1\}$ and with the following possible sequences:

1. $\{e_1, E_2\}$; $\{e_3, E_2\}$: in this case A_S becomes decomposable, without emerging new violations of decomposability conditions;

2. $\{e_1, E_2\}$; $\{e_2, E_1\}$; $\{\{e_4, e_6\}, E_1\}$; $\{e_8, E_1\}$: if after including e_1 in E_2, e_2 is included in E_1, then two $DC4$-violating tuples $(\delta(q_0, a), \varepsilon, e_4, e_2, E_1)$ and $(\delta(q_0, c), \varepsilon, e_6, e_2, E_1)$ emerge that in turn require $\{e_4, e_6\}$ to be attached to E_1. Inclusion of e_4 in E_1, on the other hand, introduces another $DC4$-violating tuple $(\delta(q_0, f), \varepsilon, e_8, e_4, E_1)$ that calls for the attachment of e_8 to E_1; similarly
3. $\{e_3, E_2\}$; $\{e_1, E_2\}$;
4. $\{e_3, E_2\}$; $\{e_2, E_1\}$; $\{\{e_4, e_6\}, E_1\}$; $\{e_8, E_1\}$, and
5. $\{e_2, E_1\}$; $\{\{e_4, e_6\}, E_1\}$; $\{e_8, E_1\}$.

In this example, the first and the third sequences, i.e., $\{\{e_1, e_3\}, E_2\}$ gives the optimal choice with the minimum number of added communication links, although initially $\{e_2, E_1\}$ sought to offer the optimal solution.

Therefore, in general an optimal solution to Problem 5.1 will be obtained through an exhaustive search, using Lemmas 5.2, 5.3 and 5.4, as state in the following algorithm.

Algorithm 5.2

1. *For any local event set, exclude any passive event whose exclusion respects* $EF1$*-*$EF4$*;*
2. *identify all* $DC1\&2$*-violating tuples,* $DC3$*-violating tuples and* $DC4$*-violating tuples and their respective enforcing tuples;*
3. *among all enforcing tuples, find the one that corresponds to the most violating tuples;*

4. *if applying of the enforcing tuple with the maximum number of violating tuples, does not impose new violations of $DC3$ or $DC4$, then apply it, go to Step 3 and continue until there is no violating tuples; otherwise, do the exhaustive search to find the sequence of link additions with the minimum number of added links.*

5. *end.*

Lemma 5.5 *Consider a deterministic task automaton A_S with local event sets E_i such that $E = \overset{n}{\underset{i=1}{\cup}} E_i$. If A_S is not decomposable with respect to parallel composition and natural projections P_i, $i = 1, ..., n$, Algorithm 5.2 optimally makes A_S decomposable, with the minimum number of communication links.*

Proof: See the proof in the Appendix. ∎

Remark 5.6 The complexity of the algorithm has the order of $O(|E|^5 \times |Q|^{n+1} \times log|Q|)$, where $|E|$, $|Q|$ and n respectively denote the size of state space, the size of global event set and the number of agents.

Remark 5.7 (Special Case: Automata with Mutual Exclusive Branches) When branches of A_S share no events (i.e. $\forall q \in Q$, $s, s' \in E^*$, $\delta(q, s)!$, $\delta(q, s')!$, $s \not< s'$, $s' \not< s$: $s \cap s' = \emptyset$), due to the definition of $DC3$ and $DC4$ in Corollary 3.2, $DC3$ and $DC4$ are trivially satisfied and moreover, since branches from any state share no event, then Algorithm 5.2 is reduced to Algorithm 5.1.

5.3.5 Feasible Solution for Task Decomposabilization

As Example 5.11 showed, the additions of communication links may successively introduce new violations of decomposability conditions, for which new links should be established. Therefore, in general an optimal solution to Problem 5.1 requires an exhaustive search, using Lemmas 5.2, 5.3 and 5.4. Moreover, checking of $DC3$ and $DC4$ is a nontrivial task, while it has to be accomplished initially as well as upon each link addition. It would be therefore very tractable if we can define a procedure to make $DC3$ and $DC4$ satisfied, without their examination. Following result takes an automaton whose $DC1$ and $DC2$ are made satisfied using Algorithm 5.1, and proposes a sufficient condition to fulfill $DC3$ and $DC4$.

Lemma 5.6 *Consider a deterministic automaton A_S, satisfying $DC1$ and $DC2$. A_S satisfies $DC3$ and $DC4$ if $\forall s_1, s_2 \in E^*$, $s_1 \nleqslant s_2$, $s_2 \nleqslant s_1$, $q, q_1, q_2 \in Q$, $\delta(q, s_1) = q_1 \neq \delta(q, s_2) = q_2$, $[\nexists e_1, e_2 \in E, e_1e_2 \leqslant s_1$, $e_2e_1 \leqslant s_2$, $\forall t \in E^*$, $\delta(q, e_1e_2t)! \Leftrightarrow \delta(q, e_2e_1t)!]$, $\exists e \in s_1 \cap s_2$, then $\forall i \in loc(e)$, $\forall e' \in \{e_1 \leqslant t_1, e_2 \leqslant t_2\}$, e' appears before e, one includes e' in E_i.*

Proof: See the proof in the Appendix. ∎

Remark 5.8 The condition in Lemma 5.6 intuitively means that for any two strings s_1, s_2 from any state q, sharing an event e, all agents who know this event e will be able to distinguish two strings, if they know the first event of each string. The ability of those agents that know this event e to distinguish strings s_1 and s_2, prevents illegal interleavings (to enforce $DC3$) and local nondeterminism (to satisfy $DC4$). The significance of this condition is that it does not require to check $DC3$ and $DC4$, instead provides a tractable (but more conservative) procedure to enforce $DC3$ and $DC4$. The expression $s_1 \not< s_2$, $s_2 \not< s_1$ in the lemma, is to exclude the pairs of strings that one of them is a substring of the other, as their product language does not exceed from the strings of A_S, provided $DC1$ and $DC2$. Moreover, the expression $[\nexists e_1, e_2 \in E, e_1e_2 \leqslant s_1,\ e_2e_1 \leqslant s_2,\ \forall t \in E^*,\ \delta(q, e_1e_2t)! \Leftrightarrow \delta(q, e_2e_1t)!]$ in this lemma excludes the pairs of strings e_1e_2t and e_2e_1t from any $q \in Q$ that have been already checked using $DC1$ and $DC2$ and do not form illegal interleaving strings, and hence do not need to include e_1 in the local event sets of e_2 and vice versa (see Example 5.12).

Combination of Lemmas 5.2 and 5.6 leads to the following algorithm as a sufficient condition to make a deterministic task automaton decomposable. Following algorithm uses Lemma 5.2 to enforce $DC1$ and $DC2$ followed by Lemma 5.6 to overcome the violations of $DC3$ and $DC4$.

Algorithm 5.3

1. *For a deterministic automaton A_S, with local event sets E_i, $i = 1, \dots, n$, $\forall E_i \in \{E_1, \dots, E_n\}$, $E_i^0 = E_i \backslash \{e \in E_i | e$ is passive in E_i and the exclusion of e from E_i does not violate $EF1$-$EF4\}$;*
2. *form the $DC1\&2$-Violating graph ; set $V^0(A_S) = V(A_S)$; $W^0(A_S) = W(A_S)$; $G^0(A_S) = (W(A_S), \Sigma)$; k=1;*
3. *among all events in the nodes in $W^{k-1}(A_S)$, find e with the maximum $|W_e^{k-1}(A_S, E_i^{k-1})|$, for all $E_i^{k-1} \in \{E_1^{k-1}, \dots, E_n^{k-1}\}$;*
4. *$E_i^k = E_i^{k-1} \cup \{e\}$; and delete all edges from e to E_i^k;*

5. *update* $W_e^k(A_S, E_i)$ *for all nodes of* $G(A_S)$*;*

6. *set* $k = k + 1$ *and go to step* (3)*, and continue, until there exist no edges;*

7. $\forall s_1, s_2 \in E^*$*,* $s_1 \nless s_2$*,* $s_2 \nless s_1$*,* $q, q_1, q_2 \in Q$*,* $\delta(q, s_1) = q_1 \neq \delta(q, s_2) = q_2$*,* $[\nexists e_1, e_2 \in E, e_1e_2 \leqslant s_1, e_2e_1 \leqslant s_2, \forall t \in E^*, \delta(q, e_1e_2t)! \Leftrightarrow \delta(q, e_2e_1t)!]$*,* $\exists e \in s_1 \cap s_2$*, then* $\forall i \in loc(e)$*,* $\forall e' \in \{e_1 \leqslant t_1, e_2 \leqslant t_2\}$*,* e' *appears before* e*, include* e' *in* E_i*.*

Based on this formulation, a solution to Problem 5.1 is given as the following theorem.

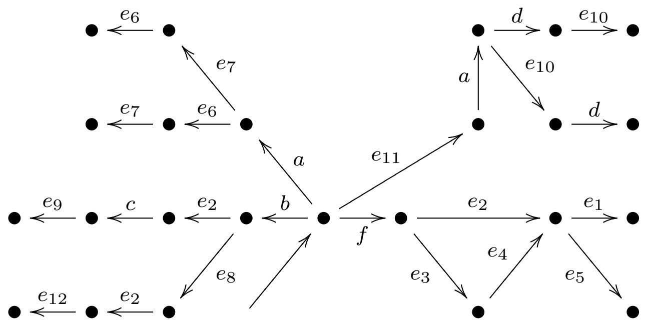
Theorem 5.1 *(Task decomposabilization) Consider a deterministic task automaton* A_S *with local event sets* E_i *such that* $E = \overset{n}{\underset{i=1}{\cup}} E_i$*. If* A_S *is not decomposable with respect to parallel composition and natural projections* P_i*,* $i = 1, ..., n$*, Algorithm 5.3 makes* A_S *decomposable. Moreover, if after Step* 6*,* $DC3$ *and* $DC4$ *are satisfied, then the algorithm makes* A_S *decomposable, with the minimum number of communication links.*

Proof: After excluding the redundant shared events in the first step, the algorithm enforces $DC1$ and $DC2$ in Steps 2 to 6, according to Lemma 5.2 and deals with $DC3$ and $DC4$ in Step 7, based on Lemma 5.6. ∎

Remark 5.9 If after Step 6, no violation of $DC3$ or $DC4$ is reported in the automaton, then A_S is made decomposable with the minimum number of added communication links; otherwise, the optimal solution can be obtained through exhaustive search by examining the number of added links for any possible sequence of enforcing tuples, using Lemmas 5.3 and 5.4, as it was presented in Lemma 5.5. To avoid the exhaustive search the algorithm provides a sufficient condition to enforce $DC3$ and $DC4$ in Step 7, according to Lemma 5.6. The complexity of the algorithm is reduced from $O(|E|^5 \times |Q|^{n+1} \times log|Q|)$ in Algorithm 5.2 to the order of $O(|E|^4 \times |Q|^3)$ in Algorithm 5.3; and also it does not anymore grow with the number of agents, all due to the elimination of enforcing of $DC3$ and $DC4$. The algorithm terminates, due to the finite number of states and events, and the fact that at the worst case, when all events are shared among all agents, the task automaton is trivially decomposable.

Example 5.12 Consider the task automaton

A_S:



with local event sets $E_1 = \{a, b, c, d, f, e_1, e_3, e_5, e_7, e_9, e_{11}\}$ and $E_2 = \{a, b, c, d, f, e_2, e_4, e_6, e_8, e_{10}, e_{12}\}$, with the communication pattern $2 \in snd_{\{a,b,c,d,f\}}(1)$ and no other communication links. This task automaton is not decomposable, due to the set of $DC1\&2$-violating tuples $\{e_1, e_2\}$, $\{e_1, e_4\}$, $\{e_2, e_3\}$, $\{e_2, e_5\}$, $\{e_3, e_4\}$, $\{e_4, e_5\}$, $DC3$-violating tuples $(e_{11}ade_{10}, ae_7e_6, a, E_1, E_2)$, $(e_{11}ade_{10}, ae_6e_7, a, E_1, E_2)$, $(e_{11}ae_{10}d, ae_7e_6, a, E_1, E_2)$, $(e_{11}ae_{10}d, ae_6e_7, a, E_1, E_2)$ and $DC4$-violating tuple $(q_0, e_{11}, \varepsilon, a, E_2)$. There is also one event d that is redundantly shared with E_2, as d is passive in E_2 and its exclusion respects $EF1$-$EF4$. Therefore, the algorithm excludes d from E_2, at the first step.

Next step is to construct the $DC1\&2$-Violating graph and remove its edges by sharing one node from each edge. The set of $DC1\&2$-Violating event pair is obtained as $V^0(A_S) = \{\{e_1, e_2\}, \{e_1, e_4\}, \{e_2, e_3\}, \{e_2, e_5\}, \{e_3, e_4\}, \{e_4, e_5\}\}$ with $W^0(A_S) = \{e_1, e_2, e_3, e_4, e_5\}$. It can be seen that the private events $d, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}$, and shared events a, b, c, f are not included in $W^0(A_S)$ as they have no contribution in the violation of $DC1$ and $DC2$. The $DC1\&2$-Violating graph is shown in Figure 5.2(a).

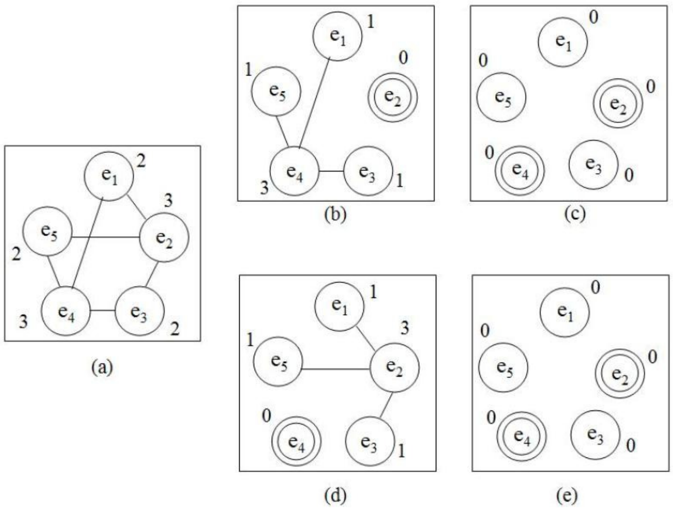


Figure 5.2: Illustration of enforcing $DC1$ and $DC2$ in Example 5.12, using Algorithm 5.3.

The maximum $|W_e^{k-1}(A_S, E_i^{k-1})|$ is formed by $\{e_2, e_4\}$ with respect to E_1 (here, since the system has only two local event sets, $|W_e^{k-1}(A_S, E_i^{k-1})|$ coincides to the valency of e in the graph). Marking e_2, including it to E_1 ($E_1^1 = \{a, b, c, d, f, e_1, e_3, e_5, e_7, e_9, e_{11}, e_2\}$), removing its incident edges to E_1 and updating the $|W_e^k(A_S, E_i^k)|$ (valencies) are shown in Figure 5.2(b). The next step will include

e_4 in E_1 ($E_1^2 = \{a, b, c, d, f, e_1, e_3, e_5, e_7, e_9, e_{11}, e_2, e_4\}$) with the highest $|W_e^k(A_S, E_i^k)|$; that removing its incident edges to E_1 and updating the $|W_e^k(A_S, E_i^k)|$ will enforce $DC1$ and $DC2$ upon Step 6, as it is illustrated in Figure 5.2 (c). If from the first stage e_4 was chosen instead of e_2, the procedure was similarly performed as depicted in Figures 5.2 (d) and (e), resulting the same set of private events $\{e_2, e_4\}$ to be shared with E_1. Inclusion of e_2 in E_1, however, introduces a new $DC4$-violating tuple $(\delta(q_0, b), \varepsilon, e_8, e_2, E_1)$ that will be automatically overcome in Step 7 by sharing $e_8 \in s_1 = e_8e_2e_{12}$ (as $s_1 = e_8e_2e_{12}$ together with $s_2 = e_2ce_9$ evolve from $\delta(q_0, b)$, sharing $e_2 \in s_1 \cap s_2$) in all local event sets of e_2, i.e., by including e_8 into E_1. Similarly, the inclusion of e_{11} in E_2 overcomes $DC4$-violating tuple $(q_0, e_{11}, \varepsilon, a, E_2)$. It is worth noting that the expression "$\nexists e_1, e_2 \in E, e_1e_2 \leqslant s_1$, $e_2e_1 \leqslant s_2$, $\forall t \in E^*$, $\delta(q, e_1e_2t)! \Leftrightarrow \delta(q, e_2e_1t)!$" in Step 7 prevents the unnecessary inclusion of e_{10} in E_1 as well as e_7 in E_2 and e_6 in E_1 (e_6 and e_7 satisfy $DC1$-$DC2$ and e_{10} and d satisfy $EF1$-$EF2$). The algorithm terminates in this stages, leading to the decomposability of A_S, with $E_1^3 = \{a, b, c, d, f, e_1, e_3, e_5, e_7, e_9, e_{11}, e_2, e_4, e_8\}$, $E_2^3 = \{a, b, c, f, e_2, e_4, e_6, e_8, e_{10}, e_{11}, e_{12}\}$.

5.4 Conclusion

The chapter proposed a method for task automaton decomposabilization, applicable in the top-down cooperative control of distributed discrete event systems. This result is a continuation of previous works on task automaton decomposability in Chapters 2 and 3, and fault-tolerant cooperative tasking in Chapter 4; and investigates the follow-up question to understand that how can an originally indecomposable task automaton be made decomposable, by modifying the event distribution among the agents.

First, using the decomposability conditions, the sources of indecomposability have been characterized and then a procedure was proposed to establish new communication links in order to enforce the decomposability conditions. To avoid the exhaustive search and the difficulty of checking of decomposability conditions in each step, a feasible solution was proposed as a sufficient condition that can make any deterministic task automaton decomposable.

5.5 Appendix

5.5.1 Proof of Lemma 5.2

Following lemma will be used during the proof.

Lemma 5.7 *Consider two non-increasing chains a_i, b_i, $i = 1, ..., N$, such that $a_1 \geq a_2 \geq ... \geq a_N > 0$, $b_1 \geq b_2 \geq ... \geq b_N > 0$. Then $\overset{N}{\underset{i=1}{\Sigma}} a_i < \overset{N}{\underset{i=1}{\Sigma}} b_i$ implies that $\exists k \in \{1, ..., N\}$ such that $a_k < b_k$.*

Proof: Suppose by contradiction that $\overset{N}{\underset{i=1}{\Sigma}} a_i < \overset{N}{\underset{i=1}{\Sigma}} b_i$, but, $\nexists k \in \{1, ..., N\}$ such that $a_k < b_k$. Then, $\forall k \in \{1, ..., N\} : a_k \geq b_k$. Therefore, since $a_k, b_k > 0, \forall k \in \{1, ..., N\}$, it results in $\overset{N}{\underset{i=1}{\Sigma}} a_i \geq \overset{N}{\underset{i=1}{\Sigma}} b_i$ which contradicts to the hypothesis, and the proof is followed.

■

Now, we prove Lemma 5.2 as follows. In each iteration k for the event e and local event set E_i with maximum $|W_e^{k-1}(A_S, E_i^{k-1})|$, all edges from e to E_i are deleted. Denoting the set of deleted edges in $k-th$ iteration by $\Delta\Sigma^k$, in each iteration k, some elements of Σ^{k-1} are moved into $\Delta\Sigma^k$ until after K iterations, there is no more elements in Σ^K to be moved into a new set. This iterative procedure leads to a partitioning of Σ by $\{\Delta\Sigma^k\}_{k=1}^K$, as $\{\Delta\Sigma^k\} \cap \{\Delta\Sigma^l\} = \emptyset$, $\forall k, l = \{1, ..., K\}, k \neq l$ and $\bigcup\limits_{k=1}^{K} \Delta\Sigma^k = \Sigma$. The latter equality leads to

$$\sum_{k=1}^{K} |\Delta\Sigma^k| = |\Sigma| \tag{5.1}$$

Now, we want to prove that

$$|\Delta\Sigma^k| = |\Delta\Sigma^k|_{max}, \forall k \in \{1, ..., K\} \Rightarrow K = K_{min} \tag{5.2}$$

Here, K is the total number of iterations that is also equal to the number of added communication links to remove violations of $DC1$ and $DC2$. In this sense, K is desired to be minimized.

The proof of (5.2) is by contradiction as follows. Suppose that $|\Delta\Sigma^k| = |\Delta\Sigma^k|_{max}$, $\forall k \in \{1, ..., K\}$, but, $K \neq K_{min}$, i.e., there exists another partitioning $\{\Delta'\Sigma^k\}_{k=1}^{K'}$, with $K' < K$ partitions, leading to

$$\sum_{k=1}^{K'} |\Delta'\Sigma^k| = |\Sigma| \tag{5.3}$$

In this case, from (5.1) and (5.3), we have

$$\sum_{k=1}^{K} |\Delta\Sigma^k| = \sum_{k=1}^{K'} |\Delta\Sigma^k| + \sum_{k=K'+1}^{K} |\Delta\Sigma^k| = \sum_{k=1}^{K'} |\Delta'\Sigma^k|. \tag{5.4}$$

Since $|\Delta\Sigma^k| > 0$, $\forall k \in \{1, ..., K\}$, then $\sum\limits_{k=K'+1}^{K} |\Delta\Sigma^k| > 0$, then, (5.4) results in

$$\sum_{k=1}^{K'} |\Delta\Sigma^k| < \sum_{k=1}^{K'} |\Delta'\Sigma^k|. \tag{5.5}$$

Moreover, since $|\Delta\Sigma^k| > 0$, $|\Delta'\Sigma^k| > 0$, $\forall k \in \{1, ..., K\}$, then (5.5) together with Lemma 5.7 imply that $\exists k \in \{1, ..., K'\} \subseteq \{1, ..., K\}$, such that $|\Delta\Sigma^k| < |\Delta'\Sigma^k|$, i.e., $\exists k \in \{1, ..., K\}$ such that $|\Delta\Sigma^k| \neq |\Delta\Sigma^k|_{max}$, which contradicts to the hypothesis; hence, (5.2) is proven. Moreover, if automaton A_S has no violations of $DC3$ and $DC4$ before and during the iterations, then the algorithm makes it decomposable with the minimum number of added communication links, since the problem of making decomposable is reduced to the optimal enforcing of $DC1$ and $DC2$.

5.5.2 Proof for Lemma 5.3

For any $DC3$-violating tuple (s_1, s_2, a, E_i, E_j), the exclusion of a from E_i or E_j, excludes a from $E_i \cap E_j$, leading to $p_{E_i \cap E_j}(s_1)$ and $p_{E_i \cap E_j}(s_2)$ do not start with a, and hence (s_1, s_2, a, E_i, E_j) will no longer act as a $DC3$-violating tuple.

For the second method in this lemma, firstly $\forall q \in Q$, $s_1, s_2 \in E^*$, $\delta(q, s_1)!$, $\delta(q, s_2)!$, $p_{E_i \cap E_j}(s_1)$ and $p_{E_i \cap E_j}(s_2)$ start with a, such that (s_1, s_2, a, E_i, E_j) is a $DC3$-violating tuple, $\exists b \in (E_i \cup E_j) \backslash (E_i \cap E_j)$ such that b appears before a in s_1 or s_2 (since A_S is deterministic and $p_{E_i \cap E_j}(s_1)$ and $p_{E_i \cap E_j}(s_2)$ start with a).

Two cases are possible, here: b appears in only one of the strings s_1 or s_2; or b appears in both strings. If b appears before a in only one of the strings, then without loss of generality, assume that b belongs to only s_1; hence, $\exists q, q_1, q_2, q_1', q_1'' \in Q_i \times Q_j$, $\omega_1, \omega_2 \in [(E_i \cup E_j) \backslash (E_i \cap E_j)]^*$, $\omega_1' \in (E_i \cup E_j)^*$, $a \in E_i \cap E_j$ such that $q_1' \in \delta_{i,j}(q, \omega_1)$, $q_1'' \in \delta_{i,j}(q_1', b)$, $q_1 \in \delta_{i,j}(q_1'', \omega_1')$, $\delta_{i,j}(q_1, a)!$, $q_2 \in \delta_{i,j}(q, \omega_2)$, $\delta_{i,j}(q_2, a)!$, where, $\delta_{i,j}$ is the transition relation in $P_i(A_S)||P_j(A_S)$. Now, due to synchronization constraint in the parallel composition, the inclusion of b in $E_i \cap E_j$ means that $([q_1'']_i, y)$ and $(x, [q_1'']_j)$ are accessible in $P_i(A_S)||P_j(A_S)$ only if $y \in [q_1'']_j$ and $x \in [q_1'']_i$, respectively. This means that $([q_1]_i, [q_2]_j)$ and $([q_2]_i, [q_1]_j)$ are not accessible in $P_i(A_S)||P_j(A_S)$; hence, $\overline{p_i(s_1)}|\overline{p_j(s_2)}$ and $\overline{p_i(s_2)}|\overline{p_j(s_1)}$ cannot evolve after a, and therefore, do not generate illegal strings out of the original strings, implying that (s_1, s_2, a, E_i, E_j) will no longer remain a $DC3$-violating tuple.

On the other hand, if b appears before a, in both strings s_1 and s_2, then $\exists q, q_1, q_2, q_1', q_1'', q_2', q_2'' \in Q_i \times Q_j$, $\omega_1, \omega_2 \in [(E_i \cup E_j)\backslash(E_i \cap E_j)]^*$, $\omega_1', \omega_2' \in (E_i \cup E_j)^*$, $a \in E_i \cap E_j$ such that $q_1' \in \delta_{i,j}(q, \omega_1)$, $q_1'' \in \delta_{i,j}(q_1', b)$, $q_1 \in \delta_{i,j}(q_1'', \omega_1')$, $\delta_{i,j}(q_1, a)!$, $q_2' \in \delta_{i,j}(q, \omega_2)$, $q_2'' \in \delta_{i,j}(q_2', b)$, $q_2 \in \delta_{i,j}(q_2'', \omega_2')$, $\delta_{i,j}(q_2, a)!$, that leads to the accessibility of $([q_1']_i, [q_2']_j)$ and $([q_2']_i, [q_1']_j)$ as well as $([q_1]_i, [q_2]_j)$ and $([q_2]_i, [q_1]_j)$ in $P_i(A_S)||P_j(A_S)$, that means that although (s_1, s_2, a, E_i, E_j) is no longer a $DC3$-violating tuple, (s_1, s_2, b, E_i, E_j) emerges as a new $DC3$-violating tuple.

In this case (when $\nexists b \in (E_i \cup E_j)\backslash(E_i \cap E_j)$ that appears before a in only one of the strings s_1 or s_2), instead of inclusion of b in $E_i \cap E_j$, if two different events e_1, e_2, $e_1 \neq e_2$ that appear before a in strings $p_{E_i \cup E_j}(s_1)$ and $p_{E_i \cup E_j}(s_1)$ are attached to $E_i \cap E_j$, it leads to $\exists q, q_1, q_2, q_3, q_4 \in Q_i \times Q_j$, $\omega_1, \omega_2, \omega_1', \omega_2' \in [(E_i \cup E_j)\backslash(E_i \cap E_j)]^*$, $e_1, e_2, a \in E_i \cap E_j$ such that $q_1 \in \delta_{i,j}(q, \omega_1 e_1)$, $q_3 \in \delta_{i,j}(q_1, \omega_1')$, $\delta_{i,j}(q_3, a)!$, $q_2 \in \delta_{i,j}(q, \omega_2 e_2)$, $q_4 \in \delta_{i,j}(q_2, \omega_2')$, $\delta_{i,j}(q_4, a)!$. Consequently, due to synchronization constraint in parallel composition, $([q_1]_i, [q]_j)$, $([q]_i, [q_1]_j)$, $([q_2]_i, [q]_j)$ and $([q]_i, [q_2]_j)$; hence, $([q_3]_i, [q_4]_j)$ and $([q_4]_i, [q_3]_j)$ are not accessible in $P_i(A_S)||P_j(A_S)$, i.e., no more $DC3$-violating tuples form on strings s_1 and s_2.

5.5.3 Proof for Lemma 5.4

For any $DC4$-violating tuple (q, t_1, t_2, e, E_i), with $q, q_1, q_2 \in Q$, $t_1, t_2 \in (E \backslash E_i)^*$, $e \in E_i$, $\delta(q, t_1) = q_1 \neq \delta(q, t_2) = q_2$, the exclusion of e from E_i leads to $p_i(e) = \varepsilon$, and $p_i(t_1 e) = p_i(t_2 e) = \varepsilon$, $[q]_i = [\delta(q_1, e)]_i = [\delta(q_2, e)]_i$; hence, (q, t_1, t_2, e, E_i) will no longer behave as a $DC4$-violating tuple. However, it should be noted that it may cause another nondeterminism on an event after e, and this event exclusion is allowed only if e is passive in E_i and the exclusion does not violate $EF1 - EF4$.

For the second method, i.e., event inclusion, if $\exists e' \in (t_1 \cup t_2) \backslash (t_1 \cap t_2)$, then without loss of generality, assume that $e' \in t_1 \backslash t_2$ such that $\exists q, q_1, q_2, q_1', q_1'' \in Q$, $t_1, t_2 \in (E \backslash E_i)^*$, $e \in E_i$, $\delta(q, t_1) = q_1 \neq \delta(q, t_2) = q_2$, $\delta(q_1', e') = q_1''$. In this case, the inclusion of e' in E_i leads to $p_i(t_1 e) = e'e$, while $p_i(t_2 e) = e$ and therefore, $[q_1]_i = [q_1'']_i \neq [q_2]_i$, i.e., in $P_i(A_S)$, t_1 and t_2 will no longer cause a nondeterminism on e from q and accordingly, (q, t_1, t_2, e, E_i) will not remain a $DC4$-violating tuple.

If however $\nexists e' \in (t_1 \cup t_2) \backslash (t_1 \cap t_2)$, i.e., $\forall e' \in (t_1 \cup t_2)$, $e' \in (t_1 \cap t_2)$, then the inclusion of any such e' generates a $DC4$-violating tuple (q, t_1, t_2, e', E_i). In this case, Lemma 5.4 suggests to take two different events that appear before e, one from t_1 and the other from t_2, and include them into E_i such that $\exists q, q_1, q_2, q_1', q_2', q_1'', q_2'' \in Q$, $e_1 \in t_1, e_2 \in t_2$, $e_1 \neq e_2$, $\delta(q, t_1) = q_1 \neq \delta(q, t_2) = q_2$, $\delta(q_1', e_1) = q_1''$, $\delta(q_2', e_2) = q_2''$. Thus, including e_1 and e_2 in E_i results in $p_i(t_1) = e_1$, $p_i(t_2) = e_2$, $[q_1]_i \in \delta_i([q]_i, t_1)$, $[q_2]_i \in \delta_i([q]_i, t_2)$, $[q_1]_i \neq [q_2]_i$ meaning that (q, t_1, t_2, e, E_i) is not a $DC4$-violating tuple anymore.

5.5.4 Proof for Lemma 5.5

The algorithm starts with excluding events from local event sets in which the events are passive and their exclusion do not violate $EF1$-$EF4$. From this stage onwards the decomposability conditions are no longer allowed to be enforced by link deletion; instead the algorithm removes the violations of decomposability conditions by establishing new communication links. Next, the algorithm applies enforcing tuples in the order of corresponding number of violating tuples. If no new violations of decomposability conditions emerge during the conducting of enforcing tuples, then the algorithm decomposes the task automaton with the minimum number of communication links, similar to the proof of Lemma 5.2. The reason is that the iterations partition the set of violating tuples, and applying of enforcing tuples (based on Lemmas 5.2, 5.3 and 5.4) with the maximum number of violating tuples in each iteration gives the maximum number of resolutions per link addition that leads to the minimum number of added communication links. The algorithm will terminate due to the finite number of states and events and at the worst case all events are shared among all agents to make the automaton decomposable.

5.5.5 Proof for Lemma 5.6

Denoting the expression , "$\forall E_i, E_j \in \{E_1, \ldots, E_n\}$, $i \neq j$, $a \in E_i \cap E_j$, $s = t_1at'_1, s' = t_2at'_2$, $p_{E_i \cap E_j}(t_1) = p_{E_i \cap E_j}(t_2) = \varepsilon$" as "A", and the expression "$\delta(q_0, \overset{n}{\underset{i=1}{|}} \overline{p_i\,(s_i)})!$ for any $\{s_1, \cdots, s_n\} \subseteq \tilde{L}(A_S)$, $\exists s, s' \in \{s_1, \cdots, s_n\}$, $s \neq s'$" as "B", the condition $DC3$ can be written as $A \Rightarrow B$. Now, if $\forall s_1, s_2 \in E^*$, $s_1 \nless s_2$, $s_2 \nless s_1$, $q, q_1, q_2 \in Q$,

$\delta(q, s_1) = q_1 \neq \delta(q, s_2) = q_2$, $[\nexists e_1, e_2 \in E, e_1e_2 \leqslant s_1, e_2e_1 \leqslant s_2, \forall t \in E^*, \delta(q, e_1e_2t)! \Leftrightarrow \delta(q, e_2e_1t)!]$, $\exists e \in s_1 \cap s_2$, any $e' \in \{e_1 \leqslant t_1, e_2 \leqslant t_2\}$, such that e' appears before e in s_1 and s_2, then e' is included in E_i, $\forall i \in loc(e)$, it follows that $\forall E_i, E_j \in \{E_1, \ldots, E_n\}$, $i \neq j$, $a \in E_i \cap E_j$, $s = t_1at'_1, s' = t_2at'_2$, $\delta(q_0, s)! \neq \delta(q_0, s')!$, $a \in s \cap s'$, then the first event of t_1 and t_2 belong to $E_i \cap E_j$, i.e., A (the antecedent of $DC3$) becomes false; hence, $A \Rightarrow B$ ($DC3$) holds true. Therefore, the procedure in Lemma 5.6 gives a sufficient condition to make $DC3$ always true.

It is similarly a sufficient condition for $DC4$ as follows. Let the expressions "$\forall i \in \{1, \ldots, n\}$, $x, x_1, x_2 \in Q_i$, $e \in E_i$, $t \in E_i^*$, $x_1 \in \delta_i(x, e)$, $x_2 \in \delta_i(x, e)$, $x_1 \neq x_2$" and "$\forall t \in E_i^* : \delta(x_1, t)! \Leftrightarrow \delta(x_2, t)!$" to be denoted as "C" and "D", respectively. In this case, $DC4$ can be expressed as $C \Rightarrow D$. Then, for a deterministic automaton A_S, if $\forall s_1, s_2 \in E^*$, $s_1 \nless s_2$, $s_2 \nless s_1$, $q, q_1, q_2 \in Q$, $\delta(q, s_1) = q_1 \neq \delta(q, s_2) = q_2$, $[\nexists e_1, e_2 \in E, e_1e_2 \leqslant s_1, e_2e_1 \leqslant s_2, \forall t \in E^*, \delta(q, e_1e_2t)! \Leftrightarrow \delta(q, e_2e_1t)!]$, $\exists e \in s_1 \cap s_2$, the first event of s_1 and s_2 are included in all local event sets that contain e, it results in $\neg C$(i.e., the antecedent of $DC4$ becomes false, and consequently, $DC4$ becomes always true). The reason is that $\forall E_i \in \{E_1, \ldots, E_n\}$, $t_1, t_2 \in E^*$, $q, q_1, q_2 \in Q$, $e \in E_i$, $\delta(q, t_1e) = q_1 \neq \delta(q, t_2e) = q_2$, we have $\neg[p_i(t_1) = p_i(t_2) = \varepsilon]$, preventing nondeterminism on e.

Chapter 6

Conclusions

This thesis proposed a method for automaton decomposition, applicable in top-down cooperative control of distributed multi-agent systems. Given a global specification as a deterministic automaton and provided the global event set as the union of local event sets for agents in a parallel distributed plant, the thesis has addressed three problems: (1) developing cooperative tasking using task decomposition, namely a divide-and-conquer method by decomposing the global task such that the fulfillment of local tasks guarantee the satisfaction of the global specification, by the team; (2) fault-tolerant cooperative tasking to identify the conditions under which a previously decomposable and achievable task remains decomposable and globally satisfied, in spite of some event failures; and (3) the design of event distribution, to make an originally indecomposable task automaton decomposable, to facilitate the proposed top-down cooperative tasking.

Motivating by these three research questions, the main contributions of the thesis can be outlined as follows:

- necessary and sufficient conditions have been developed for the decomposability of a deterministic task automaton with respect to parallel composition and natural projections into two local event sets. The approach has been then extended into a sufficient condition for an arbitrary finite number of agents, using a hierarchical algorithm, and finally, the decomposability conditions have been generalized into an arbitrary finite number of agents. These conditions intuitively mean that a task automaton is decomposable if and only if any decision on the order or selection between two events can be handled by at least one of the agents, and the cooperative perspective of the team of agents neither allows a string that is illegal in the global task nor disables a legal string of the global task automaton;

- it was shown that if a global task automaton is decomposed into local task automata for the individual agents and local supervisors exist for each agent, satisfying the local tasks, then the closed loop system of the team of agents is guaranteed to satisfy the global specification;
- a cooperative scenario has been developed and implemented by a team of three communicating robots, to show the concepts of task decomposability and cooperative tasking;
- to address the robustness of the proposed method, a notion of passivity has been introduced to characterize the redundant event failures. The passivity was found to be a necessary condition for preserving the decomposability under event failures, based on which necessary and sufficient conditions have been identified for a decomposable task automaton to remain decomposable under passive event failures;
- it was shown that under the passivity of the failed events and fault-tolerant

decomposability conditions, a previously decomposable and satisfied task automaton remains satisfied, even if some agents fail to fulfill their corresponding local tasks;

- the decomposability conditions and fault-tolerant decomposability conditions have been utilized to identify the sources of indecomposability, and then a procedure was proposed to establish new communication links in order to enforce the decomposability conditions. This procedure was shown to be a sufficient condition to make any deterministic task automaton decomposable.

This thesis may represent a promising step towards developing a top-down framework for cooperative control of multi-agent systems, as it was discussed at the very beginning. This work however is limited to the class of specifications with all states as marked states. Although in many specifications such as sequential tasking all states can be assigned to the accomplishments of stages of the task, in some applications only some of the states are marked states; therefore, the task decomposition should be developed in such a way that the collective task has the same marked states as the original task, and accordingly, the team of agents should collectively satisfy the global specification, preserving the marked states and avoiding the blocking problem. Cooperative tasking under marked states is particularly important in top-down symbolic control, where the global specification is given in linear temporal logic whose conversion to automaton imposes marked states.

Furthermore, the top-down framework in this work has been developed for basic distributed plant with all events controllable and observable. Future works could be extended to decomposability under partial controllability and observability. This would be important for the applications with limited sensing and actuation of the events.

Moreover, using the associativity property of parallel composition, it is possible to investigate the modular cooperative tasking to understand whether the composition of decomposable task automata is a decomposable task, to be used in modular cooperative tasking. The modular design allows to incorporate new tasks without redesigning of the previous controllers.

In addition, this work has set a basis for future work towards identifying the decomposability condition for nondeterministic automata that is very challenging and to our best of knowledge still is an open problem. The difficulty comes from the interleaving and synchronization of nondeterministic transitions. The proposed decomposability conditions for deterministic automata can be carefully revisited to hypothesize the decomposability of nondeterministic automata. The nondeterministic cooperative tasking will allow to address more complex specifications. On the other hand, reducing the decomposability conditions of deterministic automata to language separability would be directly applied, and afterwards the Ramadge-Wonham framework can be used for the synthesis of local supervisors such that the global language specification is collectively satisfied.

From the industrial point of view, a good starting point to apply the idea of cooperative tasking leans towards the distributed systems whose local plants are dynamically decoupled but they are coupled through the global specification, i.e., the team is supposed to achieve a global task, cooperatively. Examples on such systems include: Distributed control systems (DCS), concurrent programming and parallel computing. In distributed control systems, each local controller has access to local information only, that is the set of sensor and actuator signals form its plant and its neighbor local plants. The interconnection of each plant to its neighbors is typically through logical signals, so-called interlocks. Each interlock is indeed an event in the local event set and could be shared with the neighbors in order for cooperative tasking. For such an application, based on the proposed conditions $DC1$-$DC4$, the global specification is checked for decomposability, if it is decomposable, then using $EF1$-$EF4$, the redundant interlocks are identified and removed from the local event sets, and if the task is not decomposable, then using the Algorithms in Chapter 5, one may add new interlocks to the systems by introducing new communication links in order to make the task decomposable and implementable, cooperatively. Another application of the cooperative tasking is for concurrent programming (computation of different blocks within a computer system) and parallel computing (execution of a program in a multi-processor/multi-core system). In the first case, different subroutines of the program serve as local agents and the common switches (if-then-else) and orders (sequences) are considered as the shared events between the subroutines. For the second case, parallel computing, the share events are exchanged between the processors, rather than the subroutines, and finally in both cases, the proposed decomposability conditions can be used to render the decentralized cooperative tasking.

Another promising direction of this result could be the decomposability of timed automata, to incorporate the time information into the global specification. In this case, based on logical and temporal specifications, the occurrence time of the events can be characterized, to be used in optimal task and resource allocation in manufacturing systems, cloud computing, smart grids, transportation, parallel computing, multi-processing, robotics and distributed process control.

Bibliography

[1] F. Daneshfar and H. Bevrani, “Multi-agent systems in control engineering: a survey,” *J. Control Sci. Eng.*, vol. 2009, pp. 2–2, 2009.

[2] R. Garcia-Espallargas and G. Recatala, “Distributed agents control system, a framework for programming distributed agents,” in *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, vol. 3, 2003, pp. 2563 – 2568.

[3] P. U. Lima and L. M. Custdio, *Multi-Robot Systems, Book Series Studies in Computational Intelligence, Book Innovations in Robot, Mobility and Control.* Berlin: Springer Berlin / Heidelberg, pp. 1–64, 2005, vol. 8.

[4] “Multi-agent robot systems as distributed autonomous systems,” *Advanced Engineering Informatics*, vol. 20, no. 1, pp. 59 – 70, 2006.

[5] D. Gu and H. Hu, “Distributed network-based formation control,” *International Journal of Systems Science*, vol. 40, no. 5, pp. 539–552, 2009.

[6] W. Dong, “Distributed optimal control of multiple systems,” *International Journal of Control*, vol. 83, no. 10, pp. 2067–2079, 2010.

[7] F. Knorn, R. Stanojevic, M. Corless, and R. Shorten, “A framework for decentralised feedback connectivity control with application to sensor networks,” *International Journal of Control*, vol. 82, no. 11, pp. 2095–2114, 2009.

[8] X. Li and Y. Xi, "Distributed connected coverage control for groups of mobile agents," *International Journal of Control*, vol. 83, no. 7, pp. 1347–1363, 2010.

[9] Q. Hui, "Hybrid consensus protocols: an impulsive dynamical system approach," *International Journal of Control*, vol. 83, no. 6, pp. 1107–1116, 2010.

[10] Z. Ji, Z. Wang, H. Lin, and Z. Wang, "Controllability of multi-agent systems with time-delay in state and switching topology," *International Journal of Control*, vol. 83, no. 2, pp. 371–386, 2010.

[11] V. R. Lesser, "Cooperative multiagent systems: A personal view of the state of the art," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, pp. 133–142, 1999.

[12] P. Tabuada and G. Pappas, "Linear time logic control of discrete-time linear systems," *Automatic Control, IEEE Transactions on*, vol. 51, no. 12, pp. 1862–1877, Dec. 2006.

[13] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. Pappas, "Symbolic planning and control of robot motion [grand challenges of robotics]," *Robotics and Automation Magazine, IEEE*, vol. 14, no. 1, pp. 61–70, Mar. 2007.

[14] M. G. Hinchey, J. L. Rash, W. Truszkowski, C. Rouff, and R. Sterritt, "Autonomous and autonomic swarms," in *Software Engineering Research and Practice*, 2005, pp. 36–44.

[15] C. A. Rouff, W. F. Truszkowski, J. L. Rash, and M. G. Hinchey, *A survey of formal methods for intelligent swarms, Technical Report TM-2005-212779.* NASA Goddard Space Flight Center, Greenbelt, Maryland: NASA Goddard

Space Flight Center, 2005.

[16] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff, "Autonomous and autonomic systems: a paradigm for future space exploration missions," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 36, no. 3, pp. 279–291, May 2006.

[17] M. Kloetzer and C. Belta, "Temporal logic planning and control of robotic swarms by hierarchical abstractions," *Robotics, IEEE Transactions on*, vol. 23, no. 2, pp. 320–330, Apr. 2007.

[18] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM, 1987, pp. 25–34.

[19] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, Nov/Dec 1995, pp. 1942 – 1948.

[20] M. Dorigo and L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 53 –66, Apr. 1997.

[21] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence.* England: Oxford University Press: Oxford, 1999.

[22] R. Olfati-Saber and R. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *Automatic Control, IEEE Transactions*

on, vol. 49, no. 9, pp. 1520 – 1533, Sep. 2004.

[23] A. Jadbabaie, J. Lin, and A. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *Automatic Control, IEEE Transactions on*, vol. 48, no. 6, pp. 988–1001, Jun. 2003.

[24] H. Tanner, A. Jadbabaie, and G. Pappas, "Stable flocking of mobile agents part ii: dynamic topology," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 2, Dec. 2003, pp. 2016 – 2021.

[25] F. Xiao, L. Wang, J. Chen, and Y. Gao, "Brief paper: Finite-time formation control for multi-agent systems," *Automatica*, vol. 45, no. 11, pp. 2605–2611, 2009.

[26] S. G. Loizou and K. J. Kyriakopoulos, *Multirobot Navigation Functions I, Book Series Studies in in H. A. P Blom and J. Lygeros (Eds.), Stochastic Hybrid Systems: Theory and Safety Critical Applications.* Springer, 2006.

[27] ——, *Multirobot Navigation Functions II, Book Series Studies in in H. A. P Blom and J. Lygeros (Eds.), Stochastic Hybrid Systems: Theory and Safety Critical Applications.* Springer, 2006.

[28] H. Tanner and A. Kumar, "Towards decentralization of multi-robot navigation functions," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, Apr. 2005, pp. 4132 – 4137.

[29] J. H. Reif and H. Wang, *Social potential fields: A distributed behavioral control for autonomous robots, In K. Goldberg, D. Halperin, J. C. Latombe, R. Wilson, (Eds.),* The Algorithmic Foundations of Robotics. Boston, MA: A. K. Peters,

1995, vol. 8.

[30] T. Laue and T. Rfer, "A behavior architecture for autonomous mobile robots based on potential fields," in *8th International. Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence, Lecture Notes in Computer Science.* Springer, 2004, pp. 122–133.

[31] Z. Ji, Z. Wang, H. Lin, and Z. Wang, "Brief paper: Interconnection topologies for multi-agent coordination under leader-follower framework," *Automatica*, vol. 45, no. 12, pp. 2857–2863, 2009.

[32] T. Schouwenaars, B. DeMoor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *European control conference, ECC 2001*, 2001, pp. 2603–2608.

[33] C. Reinl and O. v. Stryk, "Optimal control of multi-vehicle-systems under communication constraints using mixed-integer linear programming," in *RoboComm '07: Proceedings of the 1st international conference on Robot communication and coordination.* Piscataway, NJ, USA: IEEE Press, 2007, pp. 1–8.

[34] Z. Cai and Z. Peng, "Cooperative coevolutionary adaptive genetic algorithm in path planning of cooperative multi-mobile robot systems," *J. Intell. Robotics Syst.*, vol. 33, no. 1, pp. 61–71, 2002.

[35] N. Carriero and D. Gelernter, "Linda in context," *Commun. ACM*, vol. 32, no. 4, pp. 444–458, 1989.

[36] G. S. Almasi, "Overview of parallel processing," *Parallel Computing*, vol. 2, no. 3, pp. 191–203, 1985.

[37] A. Konar and L. C. Jain, *Cognitive Engineering: A Distributed Approach to Machine Intelligence (Advanced Information and Knowledge Processing).* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[38] E. H. Durfee, "Distributed problem solving and planning," *Multi-agent systems and applications*, pp. 118–149, 2006.

[39] E. Semsar-Kazerooni and K. Khorasani, "Multi-agent team cooperation: A game theory approach," *Automatica*, vol. 45, no. 10, pp. 2205–2213, 2009.

[40] J. Choi, S. Oh, and R. Horowitz, "Distributed learning and cooperative control for multi-agent systems," *Automatica*, vol. 45, no. 12, pp. 2802–2814, 2009.

[41] C. C. Cheah, S. P. Hou, and J. J. E. Slotine, "Region-based shape control for a swarm of robots," *Automatica*, vol. 45, no. 10, pp. 2406–2411, 2009.

[42] P. Tabuada and G. Pappas, "From discrete specifications to hybrid control," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 4, Dec. 2003, pp. 3366–3371.

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

[44] A. Arnold, "Synchronized products of transition systems and their analysis," *Application and Theory of Petri Nets 1998*, pp. 26–27, 1998.

[45] R. Kumar and V. K. Garg, *Modeling and Control of Logical Discrete Event Systems.* Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[46] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems.* USA: Springer, 2008.

[47] "Formal modeling methodologies for control of manufacturing cells: Survey and comparison," *Journal of Manufacturing Systems*, vol. 21, no. 1, pp. 40 – 57, 2002.

[48] E. Cinlar, *Introduction to Stochastic Processes.* Englewood Cliffs, NJ: Prentice Hall, 1975.

[49] L. Kleinrock, *Queueing systems, volume I: theory.* Wiley Interscience, 1975.

[50] L. E. Holloway, B. H. Krogh, and A. Giua, "A survey of petri net methods for controlled discrete eventsystems," *Discrete Event Dynamic Systems*, vol. 7, pp. 151–190, Apr. 1997.

[51] J. L. Peterson, *Petri Net Theory and the Modeling of Systems.* Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[52] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.

[53] K. Inan and P. Varaiya, "Finitely recursive process models for discrete event systems," *Automatic Control, IEEE Transactions on*, vol. 33, no. 7, pp. 626 –639, Jul. 1988.

[54] P. Glynn, "A gsmp formalism for discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 14 –23, Jan. 1989.

[55] G. Cohen, D. Dubois, J. Quadrat, and M. Viot, "A linear-system-theoretic view

of discrete-event processes and its use for performance evaluation in manufacturing," *Automatic Control, IEEE Transactions on*, vol. 30, no. 3, pp. 210 – 220, Mar. 1985.

[56] X.-R. Cao and Y.-C. Ho, "Models of discrete event dynamic systems," *Control Systems Magazine, IEEE*, vol. 10, no. 4, pp. 69 –76, Jun. 1990.

[57] C. E. Shannon, "A symbolic analysis of relay and switching circuits," *American Institute of Electrical Engineers, Transactions of the*, vol. 57, no. 12, pp. 713 –723, Dec. 1938.

[58] M. B. Engerstedt, E. Fazzoli, and G. Pappas, "Special section on symbolic methods for complex control systems," *Automatic Control, IEEE Transactions on*, vol. 51, no. 6, pp. 921 – 923, Jun. 2006.

[59] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Temporal-logic-based reactive mission and motion planning," *Robotics, IEEE Transactions on*, vol. 25, no. 6, pp. 1370 –1381, 2009.

[60] H. Kress-Gazit and G. Pappas, "Automatic synthesis of robot controllers for tasks with locative prepositions," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, May 2010, pp. 3215 –3220.

[61] R. Gotzhein, "Temporal logic and applications: a tutorial," *Comput. Netw. ISDN Syst.*, vol. 24, no. 3, pp. 203–218, 1992.

[62] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. Pappas, "Symbolic planning and control of robot motion [grand challenges of robotics]," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 1, pp. 61–70, 2007.

[63] M. Kloetzer and C. Belta, “Control of multi-robot teams based on ltl specifications,” in *4th IFAC Conference on Management and Control of Production and Logistics (MCPL), Sibiu, Romania*, 2007, pp. 103–108.

[64] P. Tabuada, “Symbolic models for control systems,” *Acta Inf.*, vol. 43, no. 7, pp. 477–500, 2007.

[65] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas, “Discrete abstractions of hybrid systems,” *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971 –984, Jul. 2000.

[66] A. Van der Schaft, “Equivalence of dynamical systems by bisimulation,” *Automatic Control, IEEE Transactions on*, vol. 49, no. 12, pp. 2160–2172, 2004.

[67] G. Pola, A. Girard, and P. Tabuada, “Approximately bisimilar symbolic models for nonlinear control systems,” *Automatica*, vol. 44, no. 10, pp. 2508–2516, 2008.

[68] K. Cai and W. Wonham, “Supervisor localization: A top-down approach to distributed control of discrete-event systems,” *Automatic Control, IEEE Transactions on*, vol. 55, no. 3, pp. 605 –618, Mar. 2010.

[69] R. Morin, “Decompositions of asynchronous systems,” in *CONCUR ’98: Proceedings of the 9th International Conference on Concurrency Theory.* London, UK: Springer-Verlag, 1998, pp. 549–564.

[70] K. Goh and N. Shiratori, “Modularization of a specification in lotos,” in *Network Protocols, 1993. Proceedings., 1993 International Conference on*, Oct. 1993, pp. 55 –62.

[71] K. Go and N. Shiratori, “A decomposition of a formal specification: An improved constraint-oriented method,” *IEEE Trans. Softw. Eng.*, vol. 25, no. 2, pp. 258–273, 1999.

[72] S. Kiyamura, Y. Takata, and H. Seki, “Process decomposition via synchronization events and its application to counter-process decomposition,” in *PPAM*, 2003, pp. 298–305.

[73] P. Hubbard and P. Caines, “Initial investigations of hierarchical supervisory control for multi-agent systems,” in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, 1999.

[74] M. Mukund, *From global specifications to distributed implementations, in B. Caillaud, P. Darondeau, L. Lavagno (Eds.), Synthesis and Control of Discrete Event Systems.* Springer Berlin / Heidelberg, pp. 19–35, 2002.

[75] W. Zielonka, “Notes on finite asynchronous automata,” *ITA*, vol. 21, no. 2, pp. 99–135, 1987.

[76] C. Duboc, “Mixed product and asynchronous automata,” *Theor. Comput. Sci.*, vol. 48, pp. 183–199, Dec. 1986.

[77] Y. Willner and M. Heymann, “Supervisory control of concurrent discrete-event systems,” *International Journal of Control*, vol. 54, pp. 1143–1169, 1991.

[78] C. Zhou, R. Kumar, and S. Jiang, “Control of nondeterministic discrete-event systems for bisimulation equivalence,” *Automatic Control, IEEE Transactions on*, vol. 51, no. 5, pp. 754 – 765, May 2006.

[79] F. Liu, H. Lin, and Z. Dziong, "Bisimilarity control of partially observed nondeterministic discrete event systems and a test algorithm," *Automatica*, vol. 47, no. 4, pp. 782–788, 2011.

[80] M. V. Lawson, *Finite Automata, Book Series in Control Engineering, Book Handbook of Networked and Embedded Control System.* Springer Boston / Birkhuseg, 2007, vol. I.

[81] X. Koutsoukos, P. Antsaklis, J. Stiver, and M. Lemmon, "Supervisory control of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1026–1049, Jul. 2000.

[82] P. Wolper, "Constructing automata from temporal logic formulas: a tutorial," pp. 261–277, 2002.

[83] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81 –98, 1989.

[84] P. Wang and K.-Y. Cai, "Supervisory control of discrete event systems with state-dependent controllability," *Intern. J. Syst. Sci.*, vol. 40, pp. 357–366, Apr. 2009.

[85] Y. Li and W. Wonham, "Control of vector discrete-event systems. i. the base model," *Automatic Control, IEEE Transactions on*, vol. 38, no. 8, pp. 1214 –1227, 1993.

[86] ——, "Control of vector discrete-event systems. ii. controller synthesis," *Automatic Control, IEEE Transactions on*, vol. 39, no. 3, pp. 512 –531, 1994.

[87] ——, “Concurrent vector discrete-event systems,” *Automatic Control, IEEE Transactions on*, vol. 40, no. 4, pp. 628 –638, 1995.

[88] I. Romanovski and P. Caines, “On the supervisory control of multiagent product systems,” *Automatic Control, IEEE Transactions on*, vol. 51, no. 5, pp. 794 – 799, May 2006.

[89] I. Romanovski and R. Caines, “Multi-agent product systems: controllability and non-blocking properties,” in *Discrete Event Systems, 2006 8th International Workshop on*, 2006, pp. 269 –275.

[90] “On the supervisory control of multi-agent product systems: Controllability properties,” *Systems & Control Letters*, vol. 56, no. 2, pp. 113 – 121, 2007.

[91] C. Zhou, R. Kumar, and R. Sreenivas, “Decentralized modular control of concurrent discrete event systems,” in *Decision and Control, 2007 46th IEEE Conference on*, 2007, pp. 5918 –5923.

[92] P. Gohari and W. Wonham, “On the complexity of supervisory control design in the rw framework,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 30, no. 5, pp. 643 –652, 2000.

[93] K. Rohloff and S. Lafortune, “On the computational complexity of the verification of modular discrete-event systems,” in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 1, 2002, pp. 16 – 21.

[94] T.-S. Yoo and S. Lafortune, “On the computational complexity of some problems arising in partially-observed discrete-event systems,” in *American Control Conference, 2001. Proceedings of the 2001*, 2001.

[95] N. Hadj-Alouane, S. Lafortune, and F. Lin, “Think globally, communicate, act locally: online parallel/distributed supervisory control,” in *Decision and Control, 1994., Proceedings of the 33rd IEEE Conference on*, vol. 4, 1994, pp. 3661 –3666.

[96] J. Sanchez-Blanco, “Advances and trends on decentralized supervisory control of discrete event systems,” in *Electronics, Communications and Computers, 2004. CONIELECOMP 2004. 14th International Conference on*, 2004, pp. 106 – 113.

[97] A. Tsalatsanis, A. Yalcin, and K. P. Valavanis, “Automata-based supervisory controller for a mobile robot team,” in *Robotics Symposium, 2006. LARS ’06. IEEE 3rd Latin American*, 2006, pp. 53 –59.

[98] K. C. Wong and W. M. Wonham, “Modular control and coordination of discrete-event systems,” *Discrete Event Dynamic Systems*, vol. 8, no. 3, pp. 247–297, 1998.

[99] K. C. Wong, J. G. Thistle, R. P. Malhamé, and H.-H. Hoang, “Supervisory control of distributed systems: Conflict resolution,” *Discrete Event Dynamic Systems*, vol. 10, pp. 131–186, Jan. 2000.

[100] R. Hill, D. Tilbury, and S. Lafortune, “Modular supervisory control with equivalence-based conflict resolution,” in *American Control Conference, 2008*, 2008, pp. 491 –498.

[101] L. Feng and W. Wonham, “Supervisory control architecture for discrete-event systems,” *Automatic Control, IEEE Transactions on*, vol. 53, no. 6, pp. 1449

–1461, 2008.

[102] F. Lin and W. Wonham, "Decentralized control and coordination of discrete-event systems with partial observation," *Automatic Control, IEEE Transactions on*, vol. 35, no. 12, pp. 1330 –1337, 1990.

[103] S. Jiang and R. Kumar, "Decentralized control of discrete event systems with specializations to local control and concurrent systems," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 30, no. 5, pp. 653 –660, 2000.

[104] T.-S. Yoo and S. Lafortune, "New results on decentralized supervisory control of discrete-event systems," in *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, 2000.

[105] ——, "A general architecture for decentralized supervisory control of discrete-event systems," *Discrete Event Dynamic Systems*, vol. 12, no. 3, pp. 335–377, 2002.

[106] ——, "Decentralized supervisory control with conditional decisions: supervisor existence," *Automatic Control, IEEE Transactions on*, vol. 49, no. 11, pp. 1886 – 1904, 2004.

[107] ——, "Decentralized supervisory control with conditional decisions: Supervisor realization," *Automatic Control, IEEE Transactions on*, vol. 50, no. 8, pp. 1205 – 1211, 2005.

[108] H. Chakib and A. Khoumsi, "Multi-decision decentralized control of discrete event systems : Application to the *c&p* architecture," in *Discrete Event Sys-*

tems, 2008. WODES 2008. 9th International Workshop on, May 2008, pp. 480 –485.

[109] W. Qiu and R. Kumar, “Decentralized nondeterministic supervisory control of discrete event systems,” in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 1, 2004, pp. 992 –997.

[110] S.-J. Park and K.-H. Cho, “Decentralized supervisory control of nondeterministic discrete event systems: The existence condition of a robust and nonblocking supervisor,” *Automatica*, vol. 43, no. 2, pp. 377–383, 2007.

[111] “Decentralized supervisory control of discrete-event systems,” *Information Sciences*, vol. 44, no. 3, pp. 199 – 224, 1988.

[112] K. Rudie and W. Wonham, “Think globally, act locally: decentralized supervisory control,” *Automatic Control, IEEE Transactions on*, vol. 37, no. 11, pp. 1692 –1708, Nov. 1992.

[113] K. Rudie and W. M. Wonham, “Supervisory control of communicating processes,” in *Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol Specification, Testing and Verification X*. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1990, pp. 243–257.

[114] P. Kozak and W. Wonham, “Fully decentralized solutions of supervisory control problems,” *Automatic Control, IEEE Transactions on*, vol. 40, no. 12, pp. 2094 –2097, 1995.

[115] K. T. Seow, M. T. Pham, C. Ma, and M. Yokoo, “Coordination planning: Applying control synthesis methods for a class of distributed agents,” *Control*

Systems Technology, IEEE Transactions on, vol. 17, no. 2, pp. 405 –415, 2009.

[116] K. Cai and W. Wonham, "Supervisor localization: A top-down approach to distributed control of discrete-event systems," *Automatic Control, IEEE Transactions on*, vol. 55, no. 3, pp. 605 –618, 2010.

[117] B. Gaudin and H. March, "Modular supervisory control of a class of concurrent discrete event systems," in *Proceedings WODES04, Workshop on Discrete-Event Systems*, 2004, pp. 181–186.

[118] S.-H. Lee and K. Wong., "Structural decentralised control of concurrent des," *European Journal of Control*, vol. 35, pp. 1125–1134, Oct. 2002.

[119] J. Komenda and J. van Schuppen, "Optimal solutions of modular supervisory control problems with indecomposable specification languages," in *Discrete Event Systems, 2006 8th International Workshop on*, 2006, pp. 143 –148.

[120] K. Schmidt, H. Marchand, B. Gaudin, and U. Erlangen-nrnberg, "Modular and decentralized supervisory control of concurrent discrete event systems using reduced system models," in *In Workshop on Discrete Event Systems, WODES06*, 2006.

[121] I. Castellani, M. Mukund, and P. S. Thiagarajan, "Synthesizing distributed transition systems from global specification," in *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science*. London, UK: Springer-Verlag, 1999, pp. 219–231.

[122] S. Tripakis, "Undecidable problems of decentralized observation and control," in *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*,

vol. 5, 2001, pp. 4104 –4109.

[123] ——, “Decentralized observation problems,” in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC ’05. 44th IEEE Conference on*. IEEE, 2005, pp. 6 – 11.

[124] B. Jiang, H. Yang, and V. Cocquempot, “Results and perspectives on fault tolerant control for a class of hybrid systems,” *International Journal of Control*, vol. 84, no. 2, pp. 396–411, 2011.

[125] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, “Diagnosability of discrete-event systems,” *Automatic Control, IEEE Transactions on*, vol. 40, no. 9, pp. 1555 –1575, 1995.

[126] S. Takai and T. Ushio, “Reliable decentralized supervisory control of discrete event systems,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 30, no. 5, pp. 661 –667, 2000.

[127] “Reliable supervisory control for general architecture of decentralized discrete event systems,” *Automatica*, vol. 46, no. 9, pp. 1510 – 1516, 2010.

[128] F. Lin, “Robust and adaptive supervisory control of discrete event systems,” *Automatic Control, IEEE Transactions on*, vol. 38, no. 12, pp. 1848 –1852, 1993.

[129] H. Darabi, M. Jafari, and A. Buczak, “A control switching theory for supervisory control of discrete event systems,” *Robotics and Automation, IEEE Transactions on*, vol. 19, no. 1, pp. 131 – 137, 2003.

[130] K. Rohloff, “Sensor failure tolerant supervisory control,” in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC ’05. 44th IEEE Conference on*, 2005, pp. 3493 – 3498.

[131] S. Lafortune and F. Lin, “On tolerable and desirable behaviors in supervisory control of discrete event systems,” in *Decision and Control, 1990., Proceedings of the 29th IEEE Conference on*, 1990, pp. 3434 –3439.

[132] R. M. Jensen, *A survey of formal methods for intelligent swarmsDES Controller Synthesis and Fault Tolerant Control: A Survey of Recent Advances, Tech. Rep. TR-2003-40.* IT Univ. of Copenhagen, Copenhagen, Denmark: NASA Goddard Space Flight Center, 2003.

[133] Q. Wen, R. Kumar, J. Huang, and H. Liu, “A framework for fault-tolerant control of discrete event systems,” *Automatic Control, IEEE Transactions on*, vol. 53, no. 8, pp. 1839 –1849, 2008.

[134] Y. Brave and M. Heymann, “On stabilization of discrete event processes,” *Int. J. Control*, vol. 51, no. 5, pp. 1101–1117, 1990.

[135] C. M. Özveren, A. S. Willsky, and P. J. Antsaklis, “Stability and stabilizability of discrete event dynamic systems,” *J. ACM*, vol. 38, no. 3, pp. 729–751, 1991.

[136] R. Kumar, V. Garg, Marcus, and S. I. Marcus, “Language stability and stabilizability of discrete event dynamical systems,” *SIAM Journal of Control and Optimization*, vol. 31, no. 5, pp. 1294–1320, 1993.

[137] Y. Willner and M. Heymann, “Language convergence in controlled discrete-event systems,” *Automatic Control, IEEE Transactions on*, vol. 40, no. 4, pp.

616 –627, 1995.

[138] A. Saboori and S. Zad, “Fault recovery in discrete event systems,” in *Computational Intelligence Methods and Applications, 2005 ICSC Congress on*, 2005.

[139] C. Godsil and G. Royle, *Algebraic Graph Theory.* New York: Springer, 2001.

List of Publications

- Journal papers:

  1. M. Karimadini and H. Lin, “Guaranteed Global Performance Through Local Coordination,” Automatica, Vol. 47, No. 5, May 2011, Pages 890–898.
  2. M. Karimadini and H. Lin, “Fault-tolerant Cooperative Tasking for Multi-agent Systems,” International Journal of Control, Vol. 84, No. 12, December 2011, Pages 2092-2107.
  3. M. Karimadini and H. Lin, “Cooperative Tasking for Deterministic Specification Automata”, submitted for publication, 2011.
  4. M. Karimadini and H. Lin, “ Communicate Only When Necessary: Cooperative Tasking for Multi-agent Systems,” submitted for publication, 2011.

- Conference papers:

  1. M. Karimadini and H. Lin, “Reliable Task Decomposability for Cooperative Multi-agent Systems,” 30th Chinese Control Conference (CCC2011), 2011, Pages 6550–6555.
  2. M. Karimadini and H. Lin, “Decomposability of Global Tasks for Multi-

agent Systems," 49th IEEE Conference on Decision and Control (CDC 2010), Atlanta, Georgia USA, 2010, Pages 4192–4197.

3. M. Karimadini and H. Lin, "Synchronized Task Decomposition for Two Cooperative agents," IEEE International conference on Robotics, Automation and Mechatronics (RAM 2010), 2010, Pages 368–373.
4. M. Karimadini and H. Lin, "Optimal Task Automaton Decomposablization for Cooperative Control," 8th IEEE International Conference on Control & Automation (ICCA'10), 2010, Pages 2042–2047.
5. M. Karimadini, H. Lin and T.H. Lee, "Decentralized Supervisory Control: Nondeterministic Transitions Versus Deterministic Moves," IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Singapore, July 14-17, 2009, Pages 1288–1293.

- Workshops:

  1. M. Karimadini and H. Lin, "Guaranteed global accomplishment through local coordination: Cooperative tasking among multiple agents," Proc of 15th Yale Workshop on Adaptive and Learning Systems, 2011.
  2. M. Karimadini and H. Lin, "Parallel Task Decomposition for Cooperative Multi-agent systems," Technical Presentation at the 5th IEEE Control System Chapter Graduate Student Workshop in Control and Automation held on September 25, 2009 at National University of Singapore, 2009.