HIERARCHICAL AND MODULAR CONTROL OF RECONFIGURABLE MANUFACTURING SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖVÜL ARSLAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2022

Approval of the thesis:

HIERARCHICAL AND MODULAR CONTROL OF RECONFIGURABLE MANUFACTURING SYSTEMS

submitted by **ÖVÜL ARSLAN** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Klaus Werner Schmidt
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Aydan Erkmen
Electrical and Electronics Engineering, METU _____

Prof. Dr. Klaus Werner Schmidt
Electrical and Electronics Engineering, METU _____

Prof. Dr. Umut Orguner
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Mustafa Mert Ankaralı
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Ulaş Beldek
Mechatronics Engineering, Cankaya University _____

Date: 11.02.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Övül Arslan

Signature :

ABSTRACT

HIERARCHICAL AND MODULAR CONTROL OF RECONFIGURABLE MANUFACTURING SYSTEMS

Arslan, Övül
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Klaus Werner Schmidt

FEBRUARY 2022, 78 pages

Reconfigurable manufacturing systems (RMS) were introduced as a new manufacturing concept for rapidly adjusting the production capacity and functionality of manufacturing systems. Hereby, the control of RMS requires realizing each desired configuration and changing between configurations on request, whereby a suitable design approach should scale to large-scale systems.

In this thesis, we develop a controller design method for RMS that supports modular design and is scalable to RMS of large size. As the first step, we introduce the new notion of an attraction-preserving natural observer, which makes it possible to apply abstraction-based supervisor design for state attraction. Such supervisor is needed in order to move an RMS to a correct system state when performing a configuration change and starting up a new configuration. We further extend our method to the case of modular systems in order to ensure applicability to large-scale systems. Several examples demonstrate the applicability of our method.

Keywords: Reconfigurable manufacturing systems, Discrete event systems, Supervi-

sory control, Abstraction, State attraction

ÖZ

YENİDEN AYARLANABİLİR ÜRETİM SİSTEMLERİNİN HİYERARŞİK VE MODÜLER KONTROLÜ

Arslan, Övül
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Klaus Werner Schmidt

Şubat 2022 , 78 sayfa

Yeniden yapılandırılabilir üretim sistemleri (RMS), üretim sistemlerinin üretim kapasitesini ve işlevselliğini hızlı bir şekilde ayarlamak için yeni bir üretim konsepti olarak tanıtılmıştır. Bu nedenle, RMS'nin kontrolü, istenen her konfigürasyonun gerçekleştirilmesini ve istek üzerine konfigürasyonlar arasında değişiklik yapılmasını gerektirir, böylece uygun bir tasarım yaklaşımının büyük ölçekli sistemlere ölçeklenmesi gerekmektedir.

Bu tezde, RMS için modüler tasarımı destekleyen ve büyük boyutlu RMS'ye ölçeklenebilir bir kontrolcü tasarım yöntemi geliştirilmiştir. İlk adım olarak, durum çekimi (state attraction) için soyutlamaya dayalı kontrolcü tasarımının uygulanmasını mümkün kılan yeni çekim-koruyucu doğal gözlemci (attraction-preserving natural observer) kavramı tanıtılmıştır. Bir konfigürasyon değişikliği gerçekleştirirken ve yeni bir konfigürasyon başlatırken RMS'yi doğru sistem durumuna taşımak için böyle bir kontrolcüye ihtiyaç duyulmaktadır. Yöntemimiz, büyük ölçekli sistemlere uygulanabilirliği sağlamak için modüler sistemlere genişletilmiştir. Birkaç örnek ile yöntemimizin uygulanabilirliği gösterilmiştir.

Anahtar Kelimeler: Yeniden yapılandırılabilir üretim sistemleri, Ayrık olaylı sistemler, Kontrolcü tasarımı, Yeniden boyutlama, Durum çekimi

ACKNOWLEDGMENTS

First and foremost, I want to express my gratitude to my supervisor, Prof. Dr. Klaus Werner Schmidt, for his countless hours spent on this thesis as well as his invaluable assistance, guidance, and encouragement throughout my graduate studies under his supervision.

I would also like to express my gratitude to the examining committee for their suggestions and criticisms.

Finally, I would like to thank my love, Canan Budak, for being so supportive while I was working on thesis.

Without the help of my department, family, friends, and colleagues, I would not have been able to complete this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

DES	Discrete Event Systems
SCT	Supervisory Control Theory
RMS	Reconfigurable Manufacturing Systems

CHAPTER 1

INTRODUCTION

Discrete event systems (DES) are systems that have a discrete state space and whose behavior is described by the occurrence of discrete events [1,2]. DES models are generally used for human-made systems such as manufacturing systems, transportation systems, logistic systems or communication systems [3, 4]. The literature provides different representations of DES models such as finite state automata, Petri Nets and max-plus algebra [1, 5–7]. This thesis uses finite state automata to model DES.

A common point of different DES models is that they describe the behavior of a DES, which can be expressed by a formal language, that is, a set of strings. Each string is a sequence of events and represents a sequence of actions that can occur in the DES. In order to apply control to a DES, the monolithic supervisory control theory (SCT) was introduced in [8]. It considers that there is a plant model, which describes the possible behavior of a DES and a specification, which describes the desired behavior of the DES. The main idea is then to compute a supervisor (that is again represented by a finite state automaton) to ensure that the closed loop system with plant and supervisor fulfills the specification. In this context, events of a DES are classified as controllable and uncontrollable. Controllable events such as actuator events can be disabled by a supervisor, whereas uncontrollable events such as sensor events can occur any time without the possibility of disablement. That is, it is required to find a supervisor that disables controllable events of the DES model in order to fulfill the given specification. Specifically, if an uncontrollable event should not happen, this supervisor must disable a controllable event that occurs before the undesired uncontrollable event.

A well-known problem when computing supervisors for DES is the state space ex-

plosion problem. This problem is encountered when designing supervisors for large-scale DES that have many different components. Specifically, it is the case that the number of system states increases exponentially with the number of components, which makes the supervisor design infeasible. Accordingly, as an extension of the monolithic SCT, the literature provides methods for the modular and abstraction-based supervisory control [9–11]. Here, the main idea is to compute abstractions of system components in order to obtain smaller models that are suitable for computations.

When computing supervisors for manufacturing systems, it is generally desired to realize a certain flow of products through the system, while avoiding blocking situations [12]. Such blocking situations or deadlocks can for example happen if different products need to be processed by the same machine or transported by the same equipment. As an extension of traditional manufacturing systems, reconfigurable manufacturing systems (RMS) can operate in different configurations [13–16]. For example, it is possible that an RMS is able to produce different products depending on the demand. Hereby, different product types commonly require different production sequences. In the context of DES, this means that it is required to change the supervisor controlling the RMS during run-time when changing from one product type to another. In summary, the controller design for RMS requires several properties listed as follows.

- The specified normal operation of the RMS should be realized in each individual system configuration,
- A change of configuration should be implemented by completing the operation of the current configuration and then starting up the new configuration,
- The supervisor computation for RMS should be formulated in the framework of modular and abstraction-based supervisory control for application to large-scale systems.

There are different methods with different modeling frameworks for the supervisory control of RMS. Petri Net models are used in [17–22]. The implementation of supervisors for RMS is studied in [18] and [22]. The work in [18] develops a library of Petri Net components that can be used to re-write Petri Net controllers for RMS,

whereas [22] proposed a method for the PLC implementation of reconfiguration controllers. In both papers, it is assumed that the RMS is already in a suitable state to perform the reconfiguration. Methods that try to move the RMS to a suitable state for reconfiguration are presented in [17, 19–21]. A first Petri Net method for the redesign of controllers for RMS is introduced in [17]. However, there are many simplifying assumptions. Only models without uncontrollable events are considered and deadlock avoidance is not addressed in this paper. [19] considers the design of multi-mode systems that take care of both the desired operation in each configuration of an RMS and the transition between configurations. In [20], Petri Nets are used as process models for agents in a scalable agent-based RMS framework. [22] focuses on the deadlock prevention in RMS that apply re-writing when changing configurations. As a common shortcoming, none of these methods enable the computation of nonblocking supervisors for RMS that realize a given specification. Furthermore, the application of abstractions for reducing the design complexity is not taken into account by these methods.

Automata models are used in [23–28]. As the first work on this subject, [23] gives a verbal description of the tasks and possible solution procedures for changing configurations of an RMS but without a formal treatment of the subject. [24] suggests a library of RMS component models in the form of modular finite state machines. During operation of the RMS, components from this library can be selected. Although this paper allows the analysis of the correct RMS operation, there is no design method. The supervisor design for RMS is considered in [25–28]. Here, the common assumption is that each configuration of an RMS has a start state from which the configuration should start operating. That is, when performing a reconfiguration, it is required to complete the current configuration and then move the RMS to the respective start state. [25] and [26] use the concepts of optimal supervisory control [29] and state attraction [30] to compute suitable supervisors that reach the desired start state as fast as possible (with the smallest possible number of transitions). Furthermore, [27] makes it possible to impose additional language specifications when moving to the start state of a new configuration. Although the states methods support the design of supervisors for RMS, they are all formulated in the framework of monolithic supervisory control and hence not applicable for large-scale RMS. A notable exception

is the previous work in [28] that introduces the formalism for the abstraction-based supervisory control for RMS. Nevertheless, this paper only provides the necessary definitions but does not develop the required design methods.

The main aim of this thesis is to address all the requirements stated above by providing design methods for the abstraction-based supervisory control of RMS as an extension of the work in [28]. To this end, the thesis first identifies the required ideas and conditions, which include concepts from modular and abstraction-based supervisory control as well as state attraction. Specifically, the notion of a composed state attractor is employed as an extension of the monolithic definition of state attraction. In particular, a composed attractor allows multiple components of a modular system to jointly move to a desired system state. Based on this notion, the main focus of the thesis is the computation of supervisors in order to realize composed attractors. To this end, the thesis develops a new framework for the abstraction-based state attraction, which is then extended to a modular realization. As a special feature, this framework is compatible with the classical modular and abstraction-based supervisor control such that supervisors for the normal system operation and for reconfiguration can be used together. In summary, the main contributions of the thesis are

- the formulation of abstraction-based supervisory control for state attraction,
- the formulation of modular and abstraction-based supervisory control for state attraction, which can be used for the control of RMS,
- the development of supervisor design algorithms for the modular and abstraction-based supervisory control for state attraction.

The remainder of the thesis is organized as follows. Chapter 2 introduces background information about discrete event systems and supervisory control. In Chapter 3, the concept of state attraction is extended to abstraction-based state attraction. This concept is then applied to the case of composed DES in Chapter 4 and supported by examples. Conclusion and future works are given in Chapter 5.

CHAPTER 2

PRELIMINARIES

This section provides the background information for the thesis. Since the thesis work builds on different research areas within the field of discrete event systems (DES), various concepts are reviewed and the necessary notation is introduced. Section 2.1 give general background information about DES and formal languages and finite state automata are introduced as the relevant modeling tools for DES in Section 2.2 and 2.3, respectively. The classical supervisory control theory for DES is explained in Section 2.4. Since one main subject of the thesis is the usage of abstracted system models, Section 2.5 to 2.7 provide the relevant information about suitable abstractions, their properties and algorithms. Then, the concept of state attraction is described in Section 2.8 as the second main subject of the thesis. The combination of state attraction and abstraction-based supervisory control is then considered in the problem statement in Section 2.9.

2.1 Discrete Event Systems

This thesis considers systems that can be modeled in the framework of discrete event systems (DES) [1, 8]. DES models are generally used to describe human-made systems with the following distinctive properties:

- the system has a (finite or infinite) set of discrete *states*,
- the system changes its state based on the occurrence of discrete *events*,
- a state change is denoted as a *transition*.

Hereby, it holds that the system spends time in the system state, whereas transitions are assumed to occur instantaneously. Furthermore, it has to be emphasized that a DES model has to be considered as a *logical* system model that does not capture the exact timing of transitions and their associated events but rather represents their sequential order.

A simple example of a DES is a light switch that can be modeled with two states: ON and OFF. Transitions between these states occur if someone turns on or off the light. This can be associated to the events `turn_on` and `turn_off`. Together, the simple light switch can be represented by the states ON and OFF and the transitions from state ON to state OFF with event `turn_off` and from state OFF to state ON with event `turn_on`. In line with the above description, the system spends time in the states ON and OFF, whereas the transitions (turning on and off the light) occur instantaneously.

There are two common models for DES. On the one hand, *formal languages* are used to characterize the sequential behavior of DES. On the other hand, finite state automata are a modeling tool with a graphical representation that can also be used for computations with DES. The next sections describe these DES models.

2.2 Formal Languages

Formal languages are defined using an *alphabet* Σ that is the set of all events of a DES [31]. Then, the notion of a *string* can be introduced. A string s is a finite sequence of events from the alphabet Σ and the length of the string s is written as $|s|$ and denotes the number of events in the string. The empty string ϵ is a special string with no events and has the length $|\epsilon| = 0$.

Considering the example with the light switch, the alphabet is $\Sigma = \{\texttt{turn\_on}, \texttt{turn\_off}\}$ and an example string could be $s = \texttt{turn\_on}\,\texttt{turn\_off}\,\texttt{turn\_off}$. In this example, $|s| = 3$.

When modeling the behavior of DES, the concept of formal languages is used. A formal language L is a set of strings over some alphabet Σ. In this context, there are

two special languages that have to be introduced. On the one hand, $L = \emptyset$ represents the empty language that does not contain any string. Here, it has to be emphasized that $L = \emptyset \neq L' = \{\epsilon\}$ to avoid confusion. That is, $L' = \{\epsilon\}$ is not the empty language since it contains the empty string. On the other hand, $\Sigma^\star$ is defined as the language of all strings over the alphabet Σ including the empty string ϵ. In the literature, $\Sigma^\star$ is also denoted as the Kleene Closure of Σ. In line with the definition of $\Sigma^\star$ it can also be concluded that any language L over the alphabet Σ must be a subset of $\Sigma^\star$.

Returning to the example with the light switch, it is possible to start enumerating $\Sigma^\star = \{\epsilon, \texttt{turn\_on}, \texttt{turn\_off}, \texttt{turn\_on turn\_on}, \texttt{turn\_on turn\_off}, \texttt{turn\_off turn\_on}, \texttt{turn\_off turn\_off}, \ldots\}$. Here, it is clear that $\Sigma^\star$ contains an infinite number of strings and can be enumerated by iteratively writing down all strings with a certain length.

It is further possible to define different operations on strings and languages. Considering two strings $s_1, s_2 \in \Sigma^\star$, the concatenation of both strings is the string $s_1\, s_2$. Writing a string as $s = s_1\, s_2$, the substring s_1 is called a *prefix* of s and the substring s_2 is called a suffix of s. In addition, the *prefix closure* of a language L is defined as

$$\overline{L} = \{s \in \Sigma^\star | \exists u \in \Sigma^\star \text{ such that } su \in L\} \tag{2.1}$$

That is, $\overline{L}$ contains all prefixes of strings in L, which implies that $L \subseteq \overline{L}$. A language L is prefix closed if it is equal to its prefix closure $L = \overline{L}$, that is, $L = \overline{L}$.

2.3 Finite State Automata

Finite state automata are a convenient modeling tool for DES. A finite state automaton is represented by a 5-tuple

$$G = (X, \Sigma, \delta, x_0, X_m) \tag{2.2}$$

where

- X is a finite set of *states*.
- Σ is a finite set of *events*.

- $\delta : X \times \Sigma \to X$ is a *partial transition function*.
- $x_0 \in X$ is the *initial state* (state where the automata starts from).
- $X_{\mathrm{m}} \subseteq X$ is the set of *marked states* (states that the system should reach in order to complete a task).

A finite state automaton further has a graphical representation where states are represented by circles and transitions are represented by arrows between states. Transitions are labeled by their respective events, marked states are shown as a double circle and the initial state is shown by an incoming arrow.

Consider the example automaton for the light switch in Figure 2.1. Here, the set of states is $X = \{1, 2\}$ and there are two transitions represented by $\delta(1, \texttt{turn\_on}) = 2$ and $\delta(2, \texttt{turn\_off}) = 1$. Note that the transition function is partial in the sense that it is not the case that transitions for all events are defined for all states. For example, a transition with the event `turn_on` is defined at state 1, which would be written as $\delta(1, \texttt{turn\_on})!$ (the transition exists). However, there is no transition for event `turn_off` at state 1, which would be formally written as $\neg\delta(1, \texttt{turn\_off})!$ (the transition does not exist). The initial state of G is $x_0 = 1$ and G has one marked state $X_{\mathrm{m}} = \{1\}$. Semantically, state 1 is chosen as the marked state to indicate that it is desired that the light will always be turned off again.

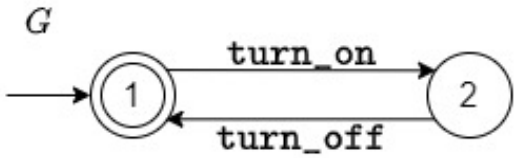

Figure 2.1: Simple automaton example.

The behavior of an automaton is represented by the languages $L(G)$ and $L_m(G)$ as follows:

$$L(G) = \{s \in \Sigma^\star | \delta(x_0, s)!\} \tag{2.3}$$

$$L_{\mathrm{m}}(G) = \{s \in L(G) | \delta(x_0, s) \in X_{\mathrm{m}}\} \tag{2.4}$$

The *closed language* $L(G)$ includes all strings that follow event sequences using the transitions starting from the initial state to any state of G. $L_{\mathrm{m}}(G)$ is the *marked*

language, which includes all strings that follow event sequences using the transitions starting from the initial state to any marked state of G.

The definition of $L(G)$ and $L_{\mathrm{m}}(G)$ implies that $L_{\mathrm{m}}(G) \subseteq L(G)$ and it is also the case that $\overline{L_{\mathrm{m}}(G)} \subseteq L(G)$. However, it is possible that there are strings $s \in L(G)$ that do not belong to $\overline{L_{\mathrm{m}}(G)}$, that is, $s \notin \overline{L_{\mathrm{m}}(G)}$. For such a string s, it is the case that s is not the prefix of a string in $L_{\mathrm{m}}(G)$, which implies that it is not possible to reach a marked state in G after s. This is denoted as a blocking situation. Accordingly, an automaton G is called *nonblocking* if the described situation does not happen. This holds if

$$\overline{L_m(G)} = L(G) \tag{2.5}$$

Large DES commonly consist of many subsystems, where each subsystem is modeled by a separate finite state automaton. The synchronized behavior of such subsystem automata can be represented by the *synchronous composition* operation. This operation synchronizes the different automata on the occurrence of their shared events, which are events that appear in the alphabets of different automata. Events that only appear in the alphabet of a single automaton are considered as *local* and hence need not be synchronized. Formally, the synchronous composition $G_1||G_2$ is defined for two automata G_1, G_2, writing $G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1}, X_{\mathrm{m},1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2}, X_{\mathrm{m},2})$. The synchronous composition is written as:

$$G_{12} = G_1||G_2 = (X_{12}, \Sigma_{12}, \delta_{12}, x_{0,12}, X_{\mathrm{m},12}) \tag{2.6}$$

with the following rules [1]:

- Set of states: $X_{12} = X_1 \times X_2$ (canonical product of X_1 and X_2),
- Alphabet: $\Sigma_{12} = \Sigma_1 \cup \Sigma_2$,
- Initial state: $x_{0,12} = (x_{0,1}, x_{0,2})$ (pair of initial states of G_1 and G_2),
- Marked states: $X_{\mathrm{m},12} = X_{\mathrm{m},1} \times X_{\mathrm{m},2}$ (canonical product of $X_{\mathrm{m},1}$ and $X_{\mathrm{m},2}$),
- Transition function: For $(x_1, x_2) \in X_{12}$ and $\sigma \in \Sigma_{12}$:

$$\delta_{12}((x_1, x_2), \sigma) = \begin{cases} (\delta_1(x_1, \sigma), \delta_2(x_2, \sigma) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \wedge \delta_1(x_1, \sigma)! \wedge \delta_2(x_2, \sigma)! \\ (\delta_1(x_1, \sigma), x_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \wedge \delta_1(x_1, \sigma)! \\ (x_1, \delta_2(x_2, \sigma)) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \wedge \delta_2(x_2, \sigma)! \end{cases}$$

That is, for shared events in $\Sigma_1 \cap \Sigma_2$, a transition is only defined at (x_1, x_2) if the event is defined at x_1 in G_1 and at x_2 in G_2 (first line). Events that are not shared (second and third line) occur independently of the respective other automaton.

As an example, consider the automata G_1 and G_2 and their synchronous composition G_{12} in Figure 2.2. It can be seen that the automata are synchronized on the shared event $\mathtt{b}$, whereas the remaining events can occur independently. In addition, it is interesting to note that, although both G_1 and G_2 are nonblocking (there is a path from any state to a marked state), this is not the case for G_{12}. In G_{12}, it is not possible to reach a marked state from $(3, 1)$.

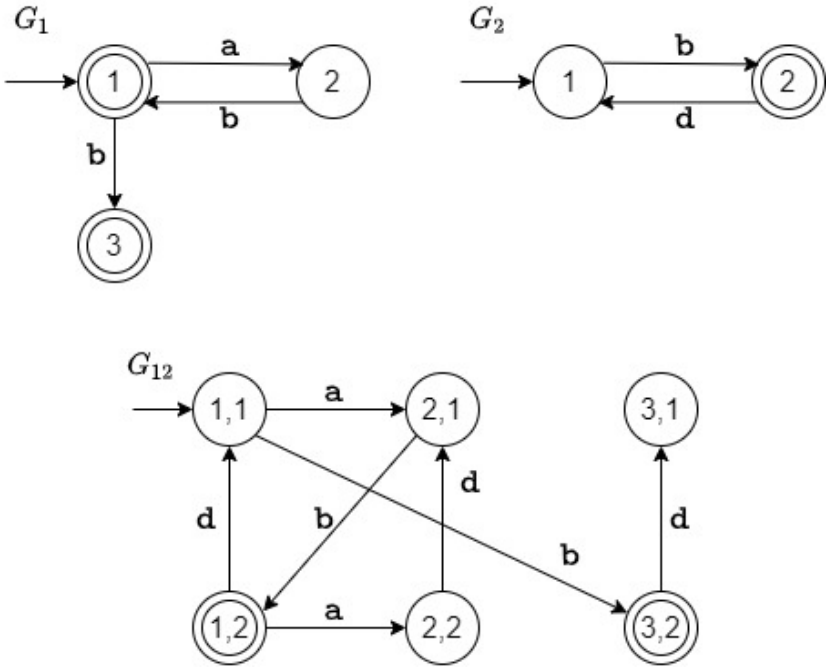

Figure 2.2: Synchronous composition example.

2.4 Supervisory Control Theory

The supervisory control theory (SCT) has the aim of controlling the behavior of a plant DES to fulfill a specification that is given in the form of a formal language. It was introduced by *Ramadge and Wonham* [8] and is hence also called the Ramadge/-Wonham framework. In order to apply control, the alphabet Σ of a DES is divided

into two disjoint subsets as follows: $\Sigma = \Sigma_{\mathrm{c}} \dot{\cup} \Sigma_{\mathrm{u}}$ and

- Σ_{c} represents the set of controllable events,
- Σ_u represents the set of uncontrollable events.

Semantically, the set Σ_{c} consists of events that can be disabled, which could for example represent actuator events. Differently, the set Σ_{u} consists of events that cannot be disabled such as sensor events.

When applying supervisory control, it is assumed that there is a system under control, denoted as the plant, whose uncontrolled behavior is modeled by an automaton G. We next introduce the notion of a *supervisor*, which can also be represented by an automaton $S = (Q, \Sigma, \nu, q_0, Q_{\mathrm{m}})$. S is a supervisor for the plant G with the uncontrollable events $\Sigma_{\mathrm{u}} \subseteq \Sigma$ if S only disables events in Σ_{c}. That is, for all $s \in L(G||S)$ and $\sigma \in \Sigma_{\mathrm{u}}$ with $s\sigma \in L(G)$ also $s\sigma \in L(S)$.

A specification is given by a language $K \subseteq L_{\mathrm{m}}(G)$ and represents desired strings. K is said to be controllable for $L(G)$ and Σ_{u} if

$$\overline{K}\Sigma_{\mathrm{u}} \cap L(G) \subseteq \overline{K}. \tag{2.7}$$

That is, each string s that is a concatenation of a desired string by an uncontrollable event ($s \in \overline{K}\Sigma_{\mathrm{u}}$) and at the same time s is possible in the plant ($s \in L(G)$) should also be allowed by the specification ($s \in \overline{K}$). The reason is that, if s was not allowed by the specification, it would not be possible to prevent s from happening since the last event of s is uncontrollable. If a specification K is controllable for $L(G)$ and Σ_{u}, it is ensured that there exists a supervisor S such that $L_{\mathrm{m}}(G||S) = K$ [8]. If K is not controllable for $L(G)$ and Σ_{u}, it is possible to compute the *supremal controllable sublanguage* of K, which is written as

$$K^{\uparrow \mathrm{c}} = \mathrm{SupC}(K, L(G), \Sigma_{\mathrm{u}}) \tag{2.8}$$

This supremal controllable sublanguage can be realized by a nonblocking supervisor in the form $L_{\mathrm{m}}(G||S) = K^{\uparrow \mathrm{c}}$ if

$$\mathrm{SupC}(K, L(G), \Sigma_{\mathrm{u}}) \neq \emptyset. \tag{2.9}$$

In this case, it is also guaranteed that $K^{\uparrow \mathrm{c}}$ represents the largest possible controllable subset of K and is hence denoted as *maximally permissive*. The closed loop system for a plant G and a supervisor S is obtained by using synchronous composition operation $G||S$. The closed language and the marked language of $G||S$ are $L(G)||L(S)$ and $L_{\mathrm{m}}(G)||L_{\mathrm{m}}(S)$.

Although specifications and controllability are formulated for languages, it is common to represent specifications by an automaton $C = (Y, \Sigma, \beta, y_0, Y_{\mathrm{m}})$ with $K = L_{\mathrm{m}}(C)$ in practice.

A supervisory control example is given in Figure 2.3. Here, it is assumed that the controllable events are shown as ticks on transitions, that is, $\Sigma_{\mathrm{c}} = \{\mathtt{a}, \mathtt{c}, \mathtt{e}, \mathtt{f}\}$ and the remaining events are uncontrollable. The specification automaton C specifies that the event $\mathtt{b}$ should not occur after the plant reaches state 3 and $\mathtt{f}$ should not occur in state 5. The specification $L_{\mathrm{m}}(C)$ is not controllable since $\mathtt{b}$ needs to be disabled at state 3 of the plant, which is not possible since $\mathtt{b} \in \Sigma_{\mathrm{u}}$. Accordingly, the supremal controllable sublanguage is computed, which is given as $L_{\mathrm{m}}(S)$ in Figure 2.3. Specifically, the event $\mathtt{c}$ has to be disable at state 2 of the plant in order to avoid reaching state 3 (where $\mathtt{b}$ could occur). In addition, $\mathtt{e}$ needs to disabled because of the same reason. Since $L_{\mathrm{m}}(S)$ is controllable for G and Σ_{u}, the automaton S can directly be used as a supervisor for G.

2.5 Natural Observer and Abstraction

The classical supervisory control theory as described above is suitable for DES of small size. In the case of large-scale DES, the problem of state space explosion is encountered, which means that such system have too many states to perform computations. This problem can partially be solved when applying abstraction-based supervisory control [9–11, 32, 33]. The main idea is to compute a smaller model of the DES that still preserves important information for the supervisor computation.

One valid approach to find such smaller model is the usage of the *natural projection* operation. Given a string $s \in \Sigma^\star$ over an alphabet Σ, the *natural projection* keeps all events in an alphabet $\hat{\Sigma} \subseteq \Sigma$ and deletes the others $(\Sigma \setminus \hat{\Sigma})$ from s. This can be

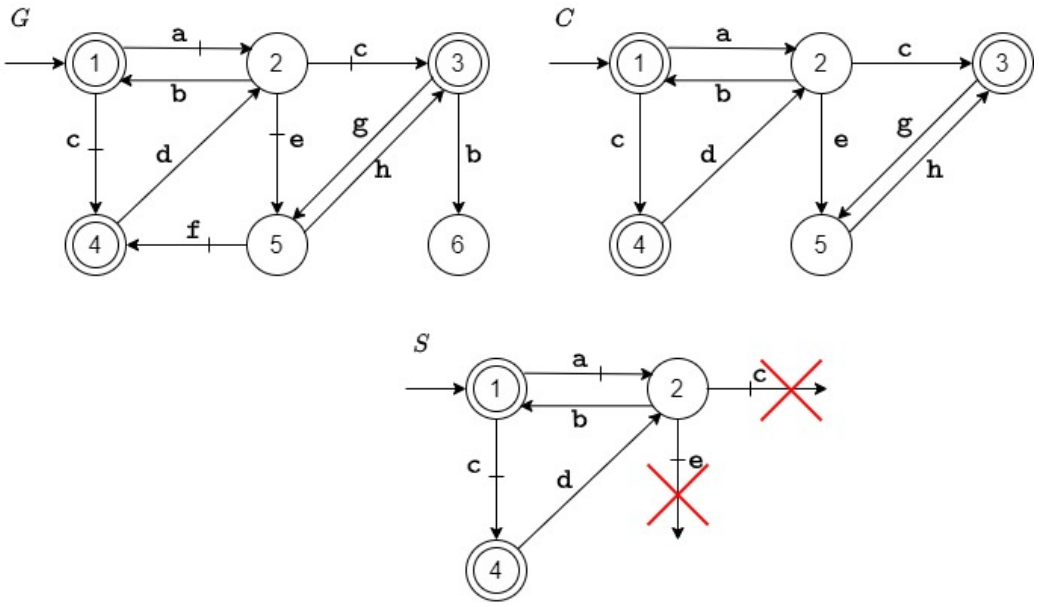

Figure 2.3: Supervisory control example.

formalized as follows. Let $\hat{\Sigma} \subseteq \Sigma$. Then, the natural projection is defined as a map $p : \Sigma^\star \to \hat{\Sigma}$ for $s \in \Sigma^\star$ and $\sigma \in \Sigma$ with

$$p(\epsilon) = \epsilon \tag{2.10}$$

$$p(\sigma) = \begin{cases} \sigma & \text{if } \sigma \in \hat{\Sigma} \\ \epsilon & \text{otherwise} \end{cases} \tag{2.11}$$

$$p(s\sigma) = p(s)p(\sigma) \tag{2.12}$$

Consider the example automaton G_1 in Figure 2.4 with the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$. Assume we choose $\hat{\Sigma} = \{\mathtt{c}, \mathtt{d}\}$. Applying the natural projection p to the example string $s = \mathtt{a\,b\,c\,d\,b} \in L_\mathrm{m}(G_1)$, we obtain $p(s) = \mathtt{c\,d}$. That is, the occurrences of the events $\mathtt{a}$ and $\mathtt{b}$, which belong to $\Sigma \setminus \hat{\Sigma}$ are erased from s.

It is further possible to define the natural projection for languages by applying the natural projection to each string of the language. That is, for $L \subseteq \Sigma^\star$, it holds that

$$p(L) = \{p(s) | s \in L\}. \tag{2.13}$$

The resulting language $p(L)$ can again be represented by an automaton. In addition, the projection can be directly applied to the languages of a given automaton G. The

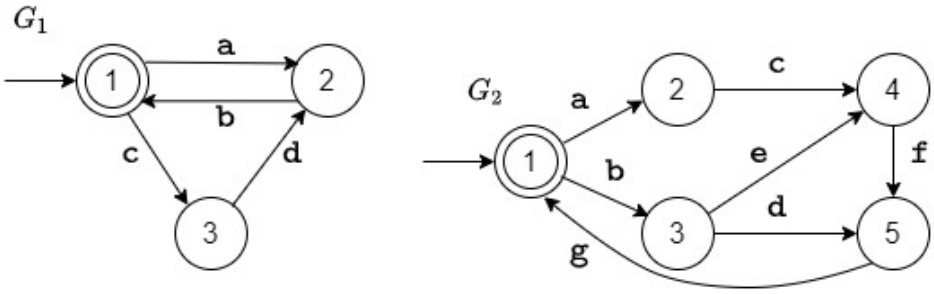

Figure 2.4: Example automata.

result is an automaton $\hat{G}$ such that

$$L_{\mathrm{m}}(\hat{G}) = p(L_{\mathrm{m}}(G)) \quad \text{and} \quad L(\hat{G}) = p(L(G)). \tag{2.14}$$

We will call $\hat{G}$ the *abstracted automaton* of G. Examples for the automata G_1 and G_2 in Figure 2.4 are shown in Figure 2.5. Here, the abstraction alphabet $\hat{\Sigma}_1 = \{\mathtt{c}, \mathtt{d}\}$ and $\hat{\Sigma}_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$ are used.

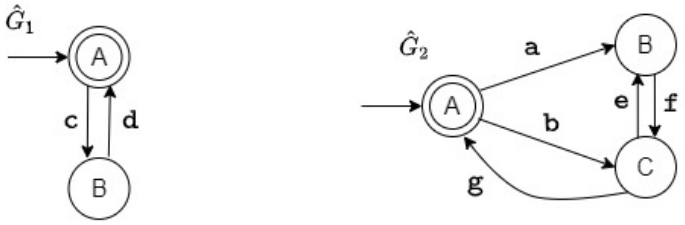

Figure 2.5: Abstraction automata.

The main goal when computing an abstraction is to obtain smaller automata for simplified computations. However, applying the natural projection is not guaranteed to produce smaller abstraction automata. In the worst case, the number of states of the abstraction automaton $\hat{G}$ can be exponential in the number of states of the original automaton G [2]. Fortunately, there are conditions that ensure that $\hat{G}$ does not have more states than G [10, 32]. The most commonly used condition for this purpose is the natural observer condition:

Definition 1 (Natural observer [32])**.** *Let $L \subseteq \Sigma^*$ be a language, and let $p_0 : \Sigma^* \to \Sigma_0^*$ be the natural projection for $\Sigma_0 \subseteq \Sigma$. p_0 is an L-observer iff for all $s \in \overline{L}$ and $t \in \Sigma_0^*$*

$$p_0(s)t \in p_0(L) \Rightarrow \exists u \in \Sigma^* \textit{ s.t. } su \in L \wedge p_0(su) = p_0(s)t.$$

In words, p_0 is an L-observer if any string $s \in \overline{L}$ can be extended to a string in L whenever its projection $p_0(s)$ can be extended to a string in $p_0(L)$. As an example, consider G_2 in Figure 2.4 and $\hat{G}_2$ in Figure 2.5 with $\hat{\Sigma}_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$. Then, the string $s = \mathtt{a} \in L(G_2)$ has the corresponding string $p(s) = \mathtt{a} \in L(\hat{G}_2)$. It can be seen that $p(s)t = \mathtt{a\,f\,g} \in L_{\mathrm{m}}(\hat{G}_2)$, that is it is possible to reach a marked state after $p(s) = \mathtt{a}$ in $\hat{G}_2$. At the same time, there is a string $u = \mathtt{c\,f\,g} \in \Sigma_2^\star$ such that $s\,u = \mathtt{a\,c\,f\,g} \in L_{\mathrm{m}}(G_2)$ and $p(u) = t$. That is, the information provided by the abstraction is correct and a marked state can also be reached in G_2. It can be verified that this condition is fulfilled for all strings in the example automata G_1 and G_2 with their respective natural projection. That is, these natural projections are natural observers. As a counter-example, we look at G_2 with a different abstraction alphabet $\hat{\Sigma}_2 = \{\mathtt{c}, \mathtt{d}, \mathtt{g}\}$. The corresponding abstraction automaton is also shown in Figure 2.6. Here, it can be checked that the natural observer condition is violated. Consider the string $s = \mathtt{a} \in L(G)$ with $p(s) = \epsilon$. Then $p(s)\,t = \epsilon p(\mathtt{d\,g}) = \mathtt{d\,g} \in L_{\mathrm{m}}(\hat{G}_2)$ but there is no string $u \in \Sigma_2^\star$ such that $\mathtt{a}u \in L_{\mathrm{m}}(G_2)$ and $p(u) = t = \mathtt{d\,g}$. In words, the abstraction $\hat{G}_2$ suggests that the sequence $\mathtt{d\,g}$ is possible after $\mathtt{a}$ but this is actually not the case in the original automaton G_2.

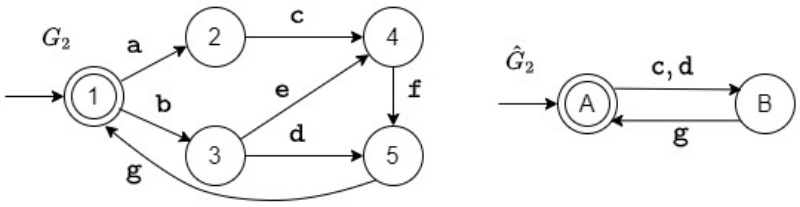

Figure 2.6: Violation of the observer condition.

The benefit of the natural observer condition is stated in Theorem 1.

Theorem 1. *Let G be an automaton over the alphabet Σ and let $\hat{\Sigma} \subseteq \Sigma$ be an abstraction alphabet. Assume the automaton for the abstraction is $\hat{G}$ such that $L_m(\hat{G}) = p(L_m(G))$. If the natural projection $p : \Sigma^* \to \hat{\Sigma}^*$ is a natural observer, then G is nonblocking if and only if $\hat{G}$ is nonblocking.*

The important implication of this theorem is that the abstraction automaton $\hat{G}$ can be used both for checking if the original automaton is nonblocking or for computing a

nonblocking supervisor as described in Section 2.4. That is, the supervisor computation can be performed with automata that have a smaller number of states.

2.6 Verification of the Natural Observer Condition

One main objective of this thesis is the computation of abstractions for a particular supervisory control problem. This computation is based on a property of the natural observer conditions that is illustrated in Figure 2.7. Specifically, it holds that the natural observer defines a partition of the state space of the original plant automaton G [34], which is illustrated for the example automata G_1 and G_2 introduced before and different abstraction alphabets. First consider G_1 in Figure 2.7. When using $\hat{\Sigma}_1 = \{\mathtt{a}\}$, it holds that all states of G_1 are in the same partition. The reason is that it is possible to reach any state from any other state in G_1 with only events in $\Sigma_1 \setminus \hat{\Sigma}_1$. When using $\hat{\Sigma}_1 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, the states of G_1 are partitioned into two classes. In the class with state 1, a marked state is reachable with only events in $\Sigma_1 \setminus \hat{\Sigma}_1$ and the abstraction events $\mathtt{a}$ and $\mathtt{c}$ are possible. Differently, in the class with the states 2 and 3, no marked state is reachable with events in $\Sigma_1 \setminus \hat{\Sigma}_1$ and only the abstraction event $\mathtt{b}$ is possible after events in $\Sigma_1 \setminus \hat{\Sigma}_1$. With the same explanation, the states are partitioned into two classes if $\hat{\Sigma}_1 = \{\mathtt{c}, \mathtt{d}\}$.

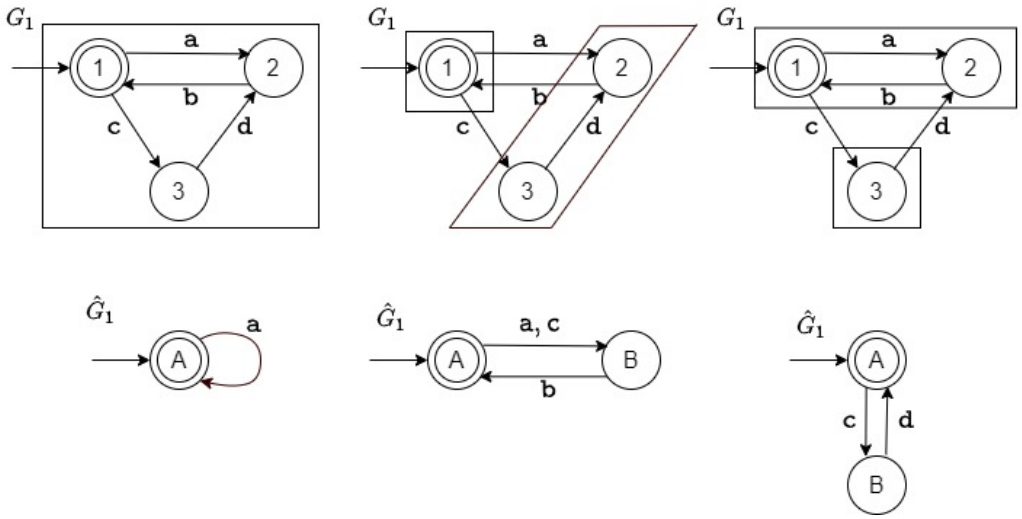

Figure 2.7: Partitions for different abstraction alphabets for G_1: $\hat{\Sigma}_1 = \{\mathtt{a}\}$ (left); $\hat{\Sigma}_1 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ (center), $\hat{\Sigma}_1 = \{\mathtt{c}, \mathtt{d}\}$ (right).

The usage of the described fact requires some theoretical background on set theory and quasi-congruences. This background is provided next with the necessary explanations.

2.6.1 Dynamic System and Quasi Congruence

We present basic results from set theory as employed in [34–36]. We denote $\mathcal{E}(M)$ the set of all *equivalence relations* on the set M. For $\mu \in \mathcal{E}(M)$, $[m]_\mu$ is the *equivalence class* containing $m \in M$. The set of equivalence classes of μ is written as $M/\mu := \{[m]_\mu | m \in M\}$ and the *canonical projection* $\mathrm{cp}_\mu : M \to M/\mu$ maps an element $m \in M$ to its equivalence class $[m]_\mu$. Let $f : M \to N$ be a function. The equivalence relation $\ker f$ is the *kernel* of f and is defined as follows: for $m, m' \in M$,

$$m \equiv m' \mod \ker f \quad \Leftrightarrow \quad f(m) = f(m'). \tag{2.15}$$

Given two equivalence relations η and μ on M, $\mu \leq \eta$, i.e. μ refines η, if $m \equiv m' \mod \mu \Rightarrow m \equiv m' \mod \eta$ for all $m, m' \in M$. In addition, we define the *meet* operation $\wedge$ for $\mathcal{E}(M)$ as follows. For any two elements $\mu, \eta \in \mathcal{E}(M)$, it holds for all $m, m' \in M$ that

$$m \equiv m' \mod (\mu \wedge \eta) \Leftrightarrow m \equiv m' \mod \mu \text{ and } m \equiv m' \mod \eta. \tag{2.16}$$

Let M and N be sets and $f : M \to 2^N$ be a set-valued function. It is also assumed that $\varphi \in \mathcal{E}(N)$, and the canonical projection cp_φ is naturally extended to sets. The equivalence relation $\varphi \circ f$ on M is defined for $m, m' \in M$ by

$$m \equiv m' \mod \varphi \circ f \Leftrightarrow \mathrm{cp}_\varphi(f(m)) = \mathrm{cp}_\varphi(f(m')). \tag{2.17}$$

Now let $f_i : M \to 2^M$ be functions, where i ranges over an index set $\mathcal{I}$. Then $S := (M, \{f_i | i \in \mathcal{I}\})$ is called a *dynamic system* [34]. The equivalence relation $\varphi \in \mathcal{E}(M)$ is called a *quasi-congruence* for S if

$$\varphi \leq \bigwedge_{i \in \mathcal{I}} (\varphi \circ f_i). \tag{2.18}$$

We next illustrate the provided notions based on the dynamic system used for the verification of the natural observer condition. Let $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$ be an automaton and let $\hat{\Sigma} \subseteq \Sigma$ with the natural projection $p : \Sigma^\star \to \hat{\Sigma}^\star$. Then, the dynamic

system $H_{\text{obs}} = (X, \{\Delta_\sigma | \sigma \in \hat{\Sigma}\} \cup \Delta_{\text{m}})$ is defined with

$$\Delta_\sigma : X \to 2^X : x \to \{\delta(x, u\sigma u') | uu' \in (\Sigma \setminus \hat{\Sigma})^*\},$$
$$\Delta_{\text{m}} : X \to 2^{X_{\text{m}}} : x \to \{\delta(x, u) \in X_{\text{m}} | u \in (\Sigma \setminus \hat{\Sigma})^*\}.$$

That is, for each abstraction event $\sigma \in \hat{\Sigma}$, a function Δ_σ is introduced. This function maps any state $x \in X$ to all states $x' \in X$ that can be reached from x with the concatenation of a substring u with events in $\Sigma \setminus \hat{\Sigma}$, the event σ and again a substring u' with events in $\Sigma \setminus \hat{\Sigma}$. Specifically, it holds that $p(u\sigma u') = \sigma$. In addition, Δ_{m} maps any state $x \in X$ to all marked states that are locally reachable, that is with a string u with events in $\Sigma \setminus \hat{\Sigma}$, which means that $p(u) = \epsilon$. The dynamic systems for the example automaton G_1 and different abstraction alphabets are shown in Figure 2.8.

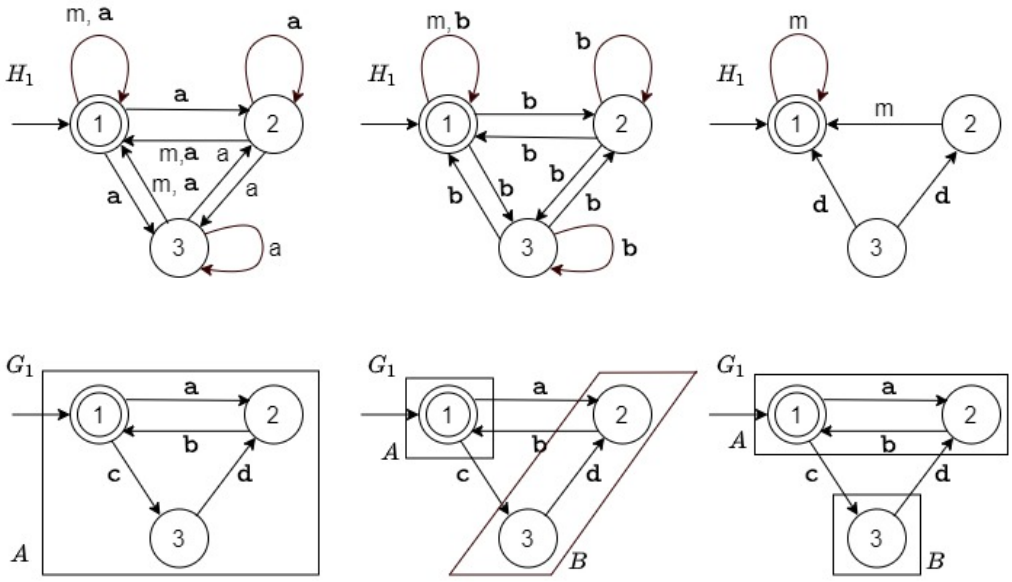

Figure 2.8: Dynamic system and quasi-congruence example: $\hat{\Sigma}_1 = \{\mathtt{a}\}$ (left), $\hat{\Sigma}_1 = \{\mathtt{b}\}$ (middle), $\hat{\Sigma}_1 = \{\mathtt{d}\}$ (right).

In order to illustrate the concept of a quasi-congruence, we look at the dynamic $H_{1,\text{obs}}$ for $\hat{\Sigma}_1 = \{\mathtt{b}\}$ in the middle of the figure. Here, the maps are as follows:

$$\Delta_{\mathtt{b}} : \begin{cases} 1 \to \{1,2,3\} \\ 2 \to \{1,2,3\} \\ 3 \to \{1,2,3\} \end{cases} \quad \text{and} \quad \Delta_{\text{m}} : \begin{cases} 1 \to \{1\} \\ 2 \to \emptyset \\ 3 \to \emptyset \end{cases} . \tag{2.19}$$

Then, a quasi-congruence for this dynamic system would be given by the equivalence relation φ with the canonical projection

$$\mathrm{cp}_\varphi : \begin{cases} 1 \to A \\ 2 \to B \\ 3 \to B \end{cases} \tag{2.20}$$

with two equivalence classes A and B. Specifically, we can determine the maps for the concatenation in (2.17) as

$$\varphi \circ \Delta_{\mathrm{b}} : \begin{cases} 1 \to \{A, B\} \\ 2 \to \{A, B\} \\ 3 \to \{A, B\} \end{cases} \quad \text{and} \quad \varphi \circ \Delta_{\mathrm{m}} : \begin{cases} 1 \to \{A\} \\ 2 \to \emptyset \\ 3 \to \emptyset \end{cases} . \tag{2.21}$$

That is, using (2.18), it holds that $1 \equiv 2 \equiv 3$ for $\varphi \circ \Delta_{\mathrm{b}}$ and $2 \equiv 3$ for $\varphi \circ \Delta_{\mathrm{m}}$. When taking the meet in (2.18) according to (2.16), we obtain that $\varphi \circ \Delta_{\mathrm{b}} \wedge \varphi \circ \Delta_{\mathrm{m}}$ induces the equivalence relation that is described by the canonical projection cp_φ introduced above. That is, indeed

$$\varphi \leq \varphi \circ \Delta_{\mathrm{b}} \wedge \varphi \circ \Delta_{\mathrm{m}}, \tag{2.22}$$

which implies that φ is a quasi-congruence for the dynamic system H_1. The associated partition of the states of G_1 is shown below the dynamic system in the figure. A similar analysis for the other examples in Figure 2.8 can be done to confirm the respective quasi-congruences.

According to [34], the coarsest quasi-congruence $\varphi^\star \in \mathcal{E}(X)$ for H_{obs} exists and can be computed with the algorithm in [37] with a complexity of $\mathcal{O}(N_X^3 \cdot N_T)$, where N_X and N_T denote the number of states and transitions of G, respectively.

2.6.2 Verification of the Natural Observer Condition

It is now possible to check the natural observer condition based on the dynamic system introduced in the previous sections. To this end, we introduce the (nondeterministic) *quotient automaton* (QA) $G_{\varphi,\hat{\Sigma}} = (Y, \hat{\Sigma} \cup \{\tau_0\}, \nu, y_0, Y_{\mathrm{m}})$ of an automaton $G = (X, \Sigma, \delta, x_0, X_{\mathrm{m}})$ for an equivalence relation $\varphi \in \mathcal{E}(X)$ and an alphabet $\hat{\Sigma} \subseteq \Sigma$ as in [34, 38]. It holds that $Y := X/\varphi$ is the *quotient set* with the associated *canonical projection* $\mathrm{cp}_\varphi : X \to Y$. The initial state and the marked states in the QA are

$y_0 = \mathrm{cp}_\varphi(x_0)$ and $Y_\mathrm{m} = \mathrm{cp}_\varphi(X_\mathrm{m})$, respectively. Also $\tau_0 \notin \Sigma$ is an additional label. The nondeterministic *transition function* $\nu : Y \times (\hat{\Sigma} \cup \{\tau_0\}) \to 2^Y$ of $G_{\varphi,\hat{\Sigma}}$ is defined as

$$\nu(y,\sigma) := \begin{cases} \{\mathrm{cp}_\varphi(\delta(x,\sigma))|x \in \mathrm{cp}_\varphi^{-1}(y)\} & \text{if } \sigma \in \hat{\Sigma} \\ \{\mathrm{cp}_\varphi(\delta(x,\gamma))|\gamma \in (\Sigma \setminus \hat{\Sigma}), x \in \mathrm{cp}_\varphi^{-1}(y)\} \setminus \{y\} & \text{if } \sigma = \tau_0. \end{cases}$$

Examples for the quasi-congruemces in Figure 2.8 are shown in Figure 2.9. The states of the quotient automata correspond to the equivalence classes of the respective quasi-congruence. In addition, transitions between the equivalence classes are introduced for events in $\hat{\Sigma}$ and for τ_0 if there are transitions with events in $\Sigma \setminus \hat{\Sigma}$ between equivalence classes. This is for example the case for the event a of the automaton in the middle of Figure 2.8.

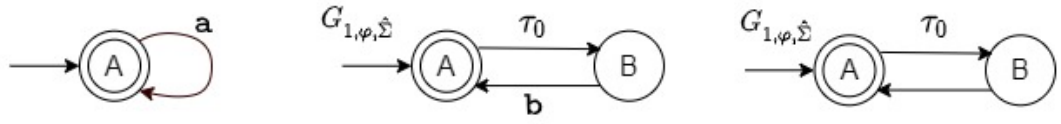

Figure 2.9: Quotient automata for the example in Figure 2.8.

It is now possible to verify the natural observer condition using the QA $G_{\varphi^\star,\hat{\Sigma}}$.

Theorem 2 (Observer Verification [34])**.** *The projection p is an $L_\mathrm{m}(G)$-observer iff $G_{\varphi^\star,\hat{\Sigma}}$ is deterministic and contains no τ_0-transitions.*

According to this theorem, the natural projection for the example on the left of Figure 2.9 is a natural observer, whereas the natural observer condition is violated for the other examples. As an additional example, Figure 2.10 shows the case where the quotient automaton is nondeterministic. Here, it is possible to reach the two different equivalence classes C and D from the state B with the same event c.

2.6.3 Computation of Abstractions

In order to apply abstraction-based supervisory control, it is desired to compute abstractions, while fulfilling the natural observer property. Hereby, it is generally the case that certain events, denoted as $\hat{\Sigma}_\cap$ must be in the abstraction alphabet, whereas

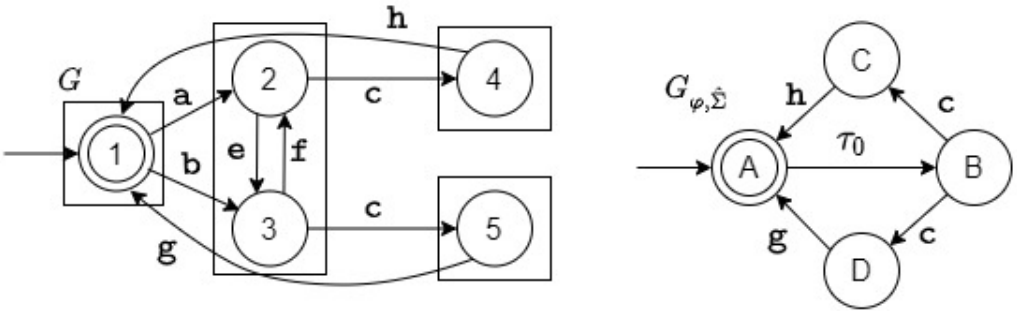

Figure 2.10: Nondeterministic quotient automaton.

the projection $p : \Sigma^\star \rightarrow \hat{\Sigma}^\star_\cap$ need not be a natural observer. In this case, it is desired to add events to the abstraction alphabet $\hat{\Sigma}$ in order to fulfill the natural observer condition.

An efficient algorithm for this purpose was proposed in [36]. Since an extension of this algorithm is developed in this thesis, we briefly describe the algorithm in [36] with the help of several examples. Consider G_1 and G_2 in Figure 2.11. Here, it is the case that the natural observer condition is violated due to τ_0 transitions in the quotient automaton. This problem can be resolved by simply adding the corresponding events to the abstraction alphabet. For G_1, $\mathtt{a}$ and $\mathtt{c}$ need to be added such that the projection with the abstraction alphabet $\hat{\Sigma}_1 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ is a natural observer. Similarly, it is enough to add $\mathtt{c}$ to obtain $\hat{\Sigma}_2 = \{\mathtt{c}, \mathtt{d}\}$ for G_2. The respective abstraction automata are also shown in the figure.

The situation is more complicated in the case of nondeterminism as can be seen in Figure 2.12. Here, the events $\mathtt{a}$ and $\mathtt{b}$ have to be added to $\hat{\Sigma}$ in order to avoid the τ_0-transition. In addition, the equivalence class corresponding to state B has two outgoing transitions with $\mathtt{c}$ to different equivalence classes. In order to make the abstraction deterministic, it is hence required to split the states of this equivalence class. This can be done by adding the events $\mathtt{e}$ and $\mathtt{f}$ to the abstraction alphabet. The main point here is that the states 4 and 5 of G are so-called exit states that have transitions in $\hat{\Sigma}$ since $\mathtt{c}$ definitely belongs to $\hat{\Sigma}$. Since the transitions with $\mathtt{c}$ introduce nondeterminism, the pair (4,5) is denoted as a bad exit state pair. Now, it is required to ensure that both states are not reachable from each other in order to split the equivalence

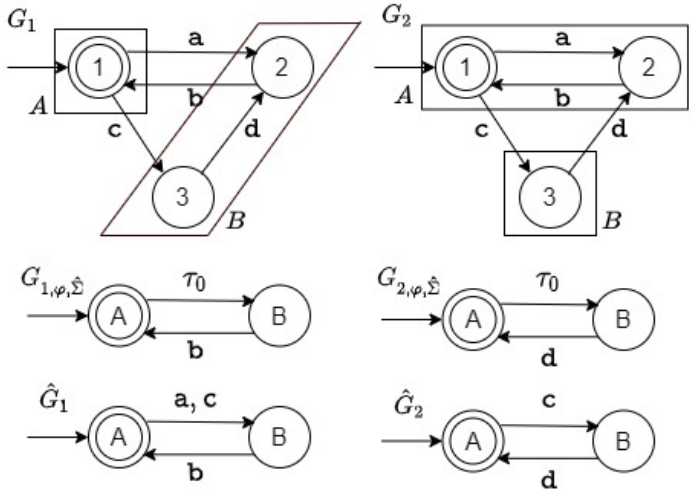

Figure 2.11: Resolving τ_0-transitions.

class. The result with the abstraction alphabet $\hat{\Sigma} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$ is also shown in the figure.

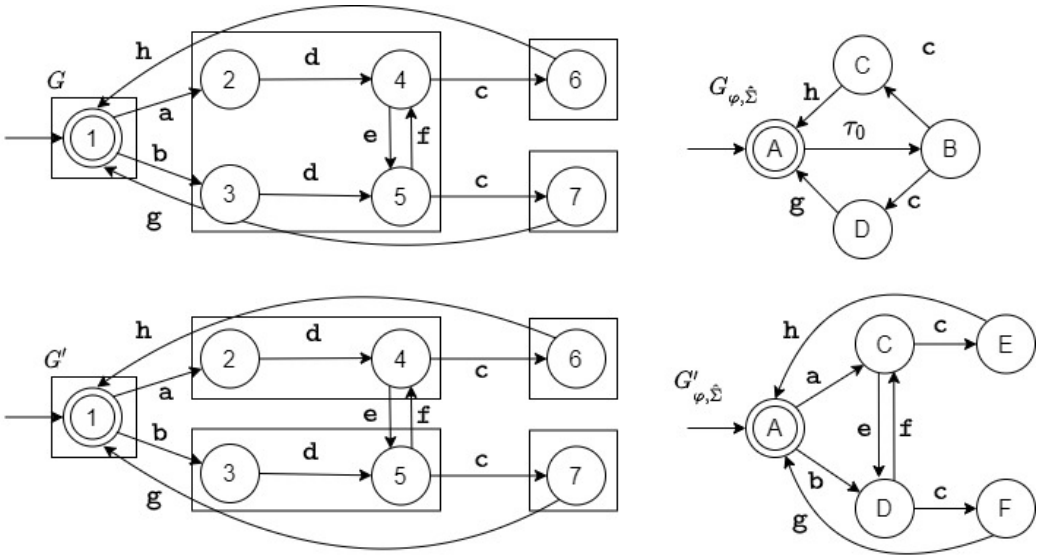

Figure 2.12: Resolving nondeterminism.

The basic algorithm for the alphabet extension according to [36] is given as follows, assuming that an automaton G and an initial abstraction alphabet $\hat{\Sigma}_\cap$ are given:

1. Initialize: $\hat{\Sigma} = \hat{\Sigma}_\cap$

2. Compute $G_{\varphi,\hat{\Sigma}}$ using the algorithm in [37]
3. Determine all events that correspond to τ_0-transitions in $G_{\varphi,\hat{\Sigma}}$. Call these events Σ_τ
4. $\hat{\Sigma} = \hat{\Sigma} \cup \Sigma_0$
5. Compute all bad exit state pairs
6. Determine all events within equivalence classes that have bad exit state pairs. Denote these events as Σ_b.
7. $\hat{\Sigma} = \hat{\Sigma} \cup \Sigma_\mathrm{b}$
8. For all events $\sigma \in \Sigma_\mathrm{b}$
   (a) Remove σ from $\hat{\Sigma}$
   (b) Check if the bad pairs are still split
   (c) If no, put σ back into $\hat{\Sigma}$
9. Check if the projection with the resulting $\hat{\Sigma}$ is a natural projection
10. If yes, the algorithm terminates
11. If no, we re-start from item 2.

According to [36], it is guaranteed that the algorithm terminates since there is only a finite number of events that can be added to $\hat{\Sigma}$. Re-starting at item 11. is required, since it is not guaranteed that the projection is a natural observer after splitting equivalence classes with bad exit state pairs.

2.7 Abstraction-based Supervisory Control

The SCT described in the previous sections is based on the assumption of a single plant automaton, a single specification and a single supervisor. Hence, this basic version of the SCT is also denoted as monolithic supervisory control. Nevertheless, it is the case in practice that DES have a modular structure with multiple components,

where each component has an individual automaton model. In addition, it is generally not the case that a specification for the overall system is given. Instead, several specifications for different components and their interaction are usually given.

We can consider the system in Figure 2.13 with the corresponding automata models in Figure 2.14 as an example of a manufacturing system that can be modeled in the DES framework. Here, there are multiple components as follows:

- SF: this component represents a stack feeder, which is used to feed products to the manufacturing system. It is modeled by an automaton with a single state and a selfloop with the event `s-r1`, which represents feeding a product to the neighboring component.
- RMT1: this component represents a reconfigurable machine tool with multiple functions, which is modeled by an automaton with two states. It is assumed that RMT1 received products from SF, then spends time in state 2 for processing and delivers products to the neighboring RT with event `r1-rt`.
- RT: this component represents a rotary table that receives products from RMT1, then rotates in direction of RMT2 and delivers products to RMT2 with the event `rt-r2`.
- RMT2: this component has the same functionality as RMT1 but is located between RT and M1. It can receive products from RT (`rt-r2`) and M1 (`m1-r2`) and deliver products to RT (`r2-rt`) and M1 (`r2-m1`).
- M1: this component represents a machine with a single function. It receives products from RMT2 (`r2-m1`) and delivers products to RMT2 (`m1-r2`).

Note that the system components are described by very simple automata in order to allow a simple representation.

In addition, we assume that the specification automata in Figure 2.15 are given for the example problem.

- D_1 specifies that products coming from SF to RMT1 should be delivered from RMT1 to RT.

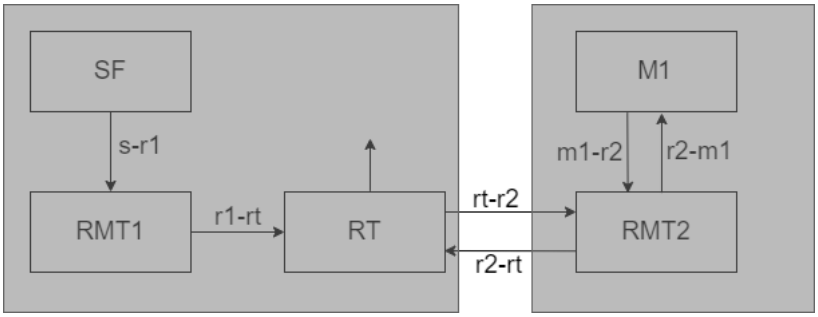

Figure 2.13: Example plant for abstraction-based control.

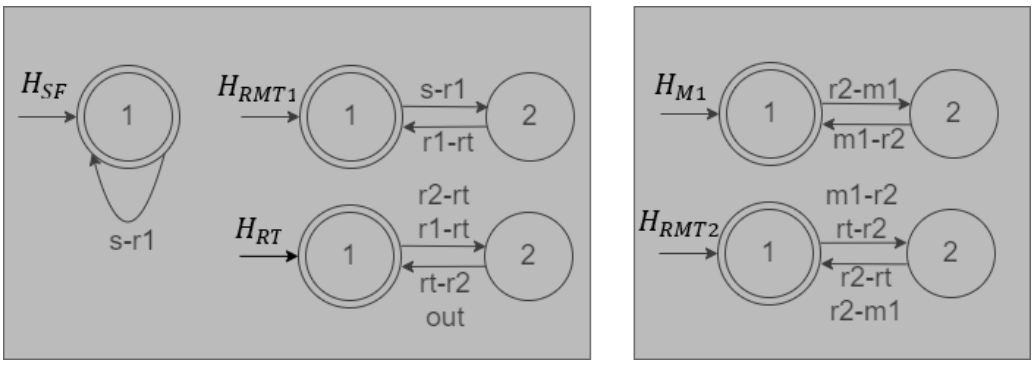

Figure 2.14: Plant automata for the example.

- D_2 specifies that products coming RMT1 to RT should be transported to RMT2.
- D_3 specifies that products coming from RT to RMT2 should be transported out of the system.
- D_4 specifies that products coming from RT to RMT2 should be transported to M1.
- D_5 specifies that products coming from RMT2 to M1 should return to RMT2 (after processing).
- D_6 specifies that products coming from M1 to RMT2 should be transported to RT.

In principle, the described scenario with multiple plant components and specifications can be converted to the monolithic case by simply computing a single plant automaton as the synchronous composition of all plant automata and a single specification as the

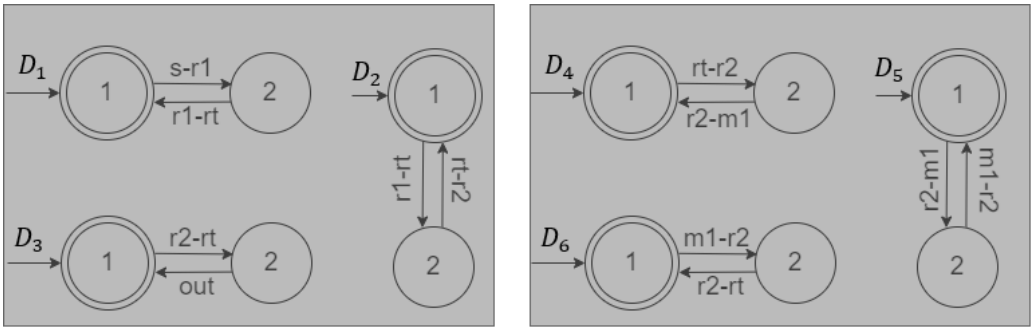

Figure 2.15: Specification automata for the example.

synchronous composition of all specification automata. However, such conversion generally leads to the so-called state space explosion problem. Specifically, it is the case that the state space of the plant automaton and specification automaton grows exponentially with the number of components. In the example, this can be seen when looking at the number of states of the plant automata, which have at most two states each. However, the number of states of the synchronous composition is in the order of the product of the component state numbers, which would be 2^4 in the example. Accordingly, it is desired to avoid computing the synchronous composition of all plant and specification automata. A method that serves this purpose is discussed next.

For the general case, we consider a DES that is modeled by multiple component automata G_i, $i = 1, \ldots, n$ with the corresponding alphabets $\Sigma_i = \Sigma_{i,\mathrm{u}} \dot{\cup} \Sigma_{i,\mathrm{c}}$. We write $\Sigma_{i,\mathrm{u}}$ and $\Sigma_{i,\mathrm{c}}$ for the uncontrollable and the controllable events, respectively. Then, it is possible to define the shared events $\Sigma_{i,\cap} = \bigcup_{k=1,k\neq i}^{n} (\Sigma_i \cap \Sigma_k)$ of each component. $\Sigma_{i,\cap}$ represents all the events that belong to Σ_i and at least one of the other component alphabets. Then, the overall set of *shared events* is given by $\Sigma_\cap = \bigcup_{i=1}^{n} \Sigma_{i,\mathrm{s}}$.

In the scope of the monolithic SCT, the overall system model is $G = \|_{i=1}^{n} G_i$ with the alphabet $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$. Considering the shared events, it is reasonable to assume that the controllability property of each shared event is unique. That is, if two components share an event, then they should agree on the control status of this event, i.e. $\forall i, k,\ i \neq k,\ \Sigma_{i,\mathrm{u}} \cap \Sigma_{k,\mathrm{c}} = \emptyset$. If this condition is fulfilled, we can define the overall sets of uncontrollable and controllable events as $\Sigma_\mathrm{u} = \bigcup_{i=1}^{n} \Sigma_{i,\mathrm{u}}$ and $\Sigma_\mathrm{c} = \bigcup_i \Sigma_{i,\mathrm{c}}$.

As noted above, also multiple specifications are usually provided. We assume that

component specifications $K_i \subseteq L_{\mathrm{m}}(G_i)$, $i = 1, \ldots, n$ are given. In addition, there can be a global specification $\hat{K} \subseteq \hat{\Sigma}^{\star}$, whereby it is assumed that $\Sigma_{\cap} \subseteq \hat{\Sigma} \subseteq \Sigma$. That is, $\hat{K}$ specifies the interaction behavior of the different plant components, which has to be formulated using an alphabet that includes the shared events $\Sigma_{\cap}$. Then, the main idea is to apply an abstraction-based approach as illustrated in Fig. 2.16 [38].

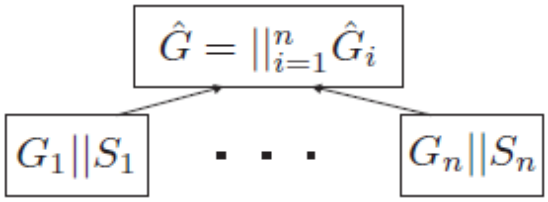

Figure 2.16: Hierarchical and decentralized control architecture.

Having the plant components G_i and the specifications K_i for $i = 1, \ldots, n$, the first step is to compute component supervisors S_i, $i = 1, \ldots, n$ by applying the standard supervisor computation as described in Section 2.4:

$$L_{\mathrm{m}}(S_i) = SupC(K_i, L(G_i), \Sigma_{\mathrm{u},i}).$$

The obtained closed loops represented by $G_i||S_i$ are then abstracted using projections $p_i : \Sigma_i^{\star} \rightarrow (\Sigma_i \cap \hat{\Sigma})^{\star}$ in order to obtain abstracted closed loops $\hat{G}_i$, $i = 1, \ldots, n$. Hereby, the automata $\hat{G}_i$ are computed such that

$$L_{\mathrm{m}}(\hat{G}_i) = p_i(L_{\mathrm{m}}(G_i||S_i)) \quad \text{and} \quad L(\hat{G}_i) = p_i(L(G_i||S_i)).$$

In the next step, it is possible to compute the overall abstracted closed loop as the synchronous composition of the abstracted closed-loops with

$$\hat{G} = ||_{i=1}^{n} \hat{G}_i$$

Noting that the alphabet of $\hat{G}$ is the same as the alphabet of the specification $\hat{K}$, it is now possible to determine an abstraction-based supervisor $\hat{S}$ such that

$$L_{\mathrm{m}}(\hat{S}) = SupC(\hat{K}, L(\hat{G}), \Sigma_{\mathrm{u}} \cap \hat{\Sigma}).$$

Together, we obtained the supervisors S_i for $i = 1, \ldots, n$ for the modular plant components and the supervisor $\hat{S}$ for the coordination of the interaction behavior of the different components. That is, we can realize the overall closed loop as

$$\hat{S}||(||_{i=1}^{n} S_i||G_i). \tag{2.23}$$

The main important fact is that the overall closed loop in (2.23) is nonblocking if the used projections p_i, $i = 1, \ldots, n$ are all natural observers as shown in [9, 38].

Applying the described abstraction-based control method to the example system, we first identify the component models. One possibility is to use all plant components as separate models, which generally leads to an inefficient design. In this example, we use two component models by grouping plant models as follows:

- $G_1 = H_{\mathrm{SF}} || H_{\mathrm{RMT1}} || H_{\mathrm{RT}}$,
- $G_2 = H_{\mathrm{RMT2}} || H_{\mathrm{M1}}$.

Then, the corresponding specifications can be written as

- $C_1 = D_1 || D_2 || D_3$ and $K_1 = L_{\mathrm{m}}(C_1)$,
- $C_2 = D_4 || D_5 || D_6$ and $K_2 = L_{\mathrm{m}}(C_2)$.

Computing $SupC(K_1, L(G_1), \Sigma_{1,\mathrm{u}}) = L_{\mathrm{m}}(S_1)$ and $SupC(K_2, L(G_2), \Sigma_{2,\mathrm{u}}) = L_{\mathrm{m}}(S_2)$, the supervisors S_1 and S_2 in Figure 2.17 for the modular components G_1 and G_2 are obtained. Here, the the shared events are $\Sigma_{\cap} = \{\texttt{r2-rt}, \texttt{rt-r2}\}$. In order to fulfill the natural observer conditions for all abstractions, also the events `s-r1` and `r1-rt` need to be added to the overall abstraction alphabet to obtain $\hat{\Sigma} = \{\texttt{s-r1}, \texttt{r1-rt}, \texttt{rt-r2}, \texttt{r2-rt}\}$. The resulting abstracted closed loops $\hat{S}_1$ and $\hat{S}_2$ are shown in Figure 2.18 together with the overall abstracted closed loop $\hat{G} = \hat{S}_1 || \hat{S}_2$.

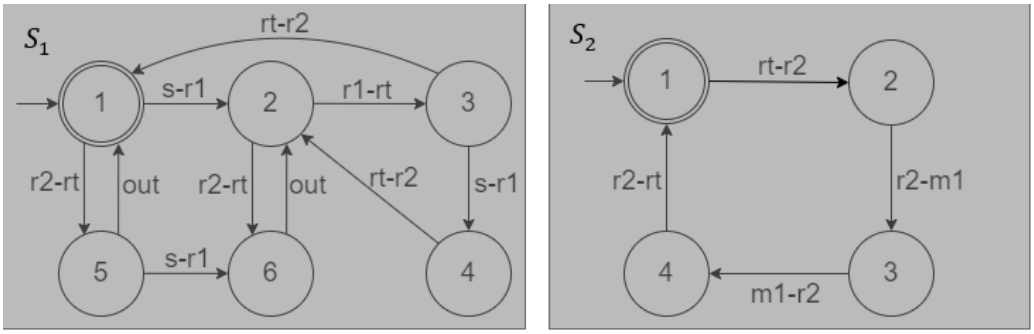

Figure 2.17: Low-level supervisors for the example.

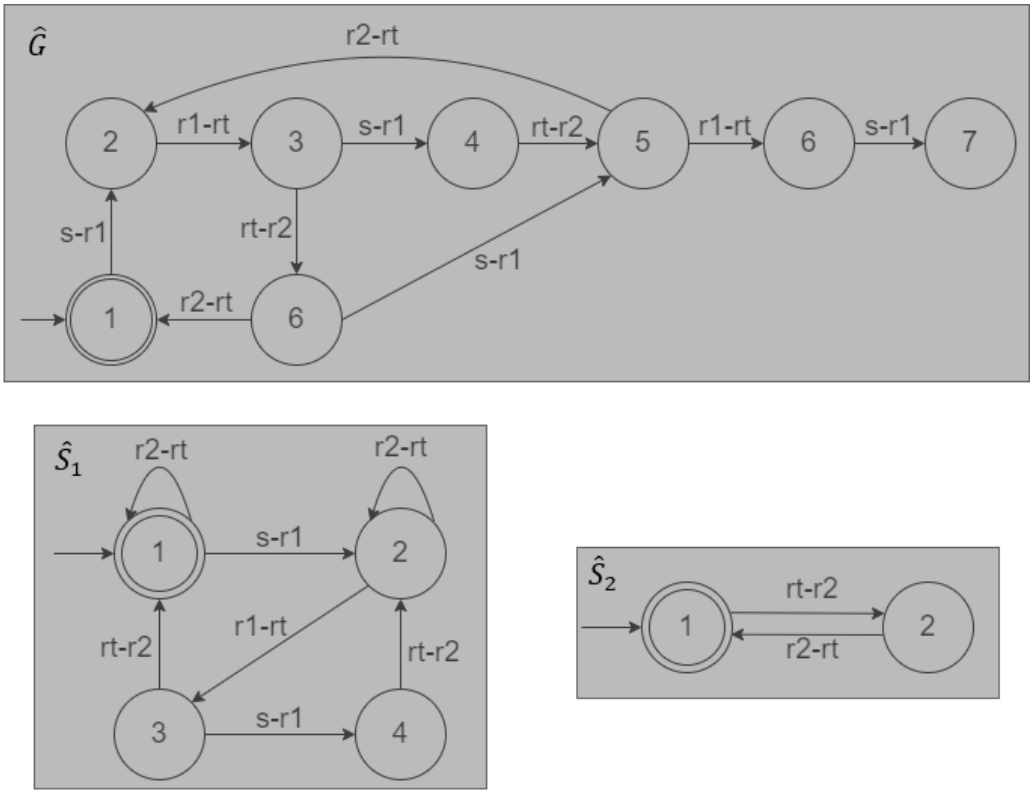

Figure 2.18: Abstracted closed loops $\hat{S}_1$ and $\hat{S}_2$ and overall abstracted closed loop $\hat{G}$.

In this example, we see that the overall abstracted closed loop is nonblocking. In addition, we want to apply a high-level specification $\hat{C}$ that is given by the automaton in Figure 2.19. It specified that the events `s-r1` and `rt-r2` should always occur alternately. Computing the corresponding high-level supervisor as $SupC(\hat{K}, L(\hat{G}), \hat{\Sigma}_{\mathrm{u}}) = L_{\mathrm{m}}(\hat{S})$, the result in Figure 2.20 is obtained. Then, the overall closed loop for this example is given by

$$\hat{S}||S_1||S_2||G_1||G_2.$$

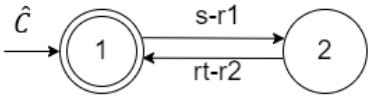

Figure 2.19: High-level specification $\hat{C}$.

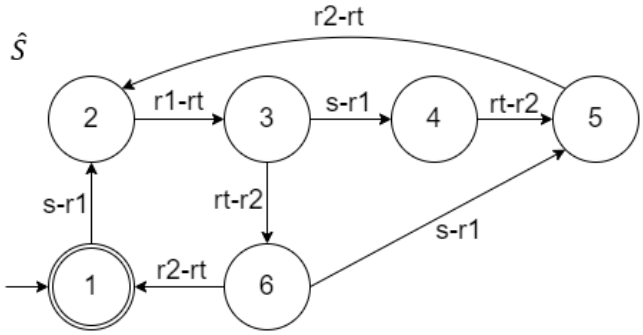

Figure 2.20: High-level supervisor $\hat{S}$.

2.8 State Attraction

Consider that $G = (X, \Sigma, \delta, x_0, X_{\mathrm{m}})$ and $G' = (X', \Sigma, \delta', x'_0, X'_{\mathrm{m}})$ are finite state automata. G' is a *subautomaton* of G, if $X' \subseteq X$, $x'_0 = x_0$ and for all $x \in X'$ and $\sigma \in \Sigma$, it holds that $\delta'(x, \sigma)! \Rightarrow \delta'(x, \sigma) = \delta(x, \sigma)$. In words, the automaton (G') is extracted from G by removing states and transitions. Then, we write ($G' \sqsubseteq G$) if (G') is a subautomaton of G. G' is a *strict subautomaton* of G if additionally $\delta(x, \sigma) \in X' \Rightarrow \delta'(x, \sigma) = \delta(x, \sigma)$. In words, only states are removed from G to obtain G'. Figure 2.21 shows an example automaton G and Figure 2.22 shows an example of a subautomaton of G (in the left) and a strict subautomaton of G (in the right).

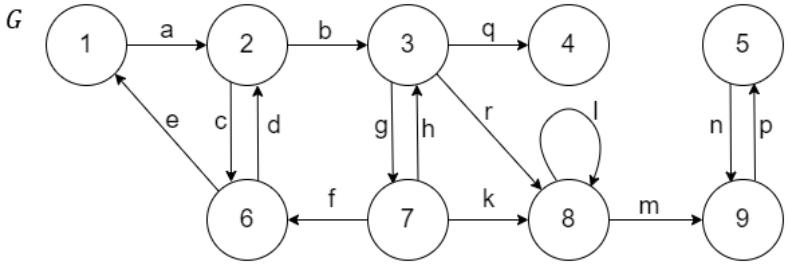

Figure 2.21: Example automaton G.

Consider an automaton $G = (X, \Sigma, \delta, x_0, X_m)$ and the uncontrollable events (Σ_u). Then, the subset $X' \subseteq X$ is denoted as invariant set in G if there is no transition

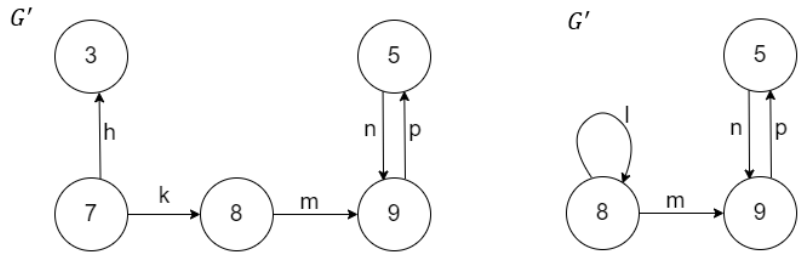

Figure 2.22: Subautomaton (left) and strict subautomaton (right) of G.

leaving the states in the subset X': [30]

$$\forall x \in X' \text{ and } \sigma \in \Sigma \text{ it must hold that } \delta(x,\sigma)! \Rightarrow \delta(x,\sigma) \in X' \tag{2.24}$$

Moreover, the set ($X' \subseteq X$) is denoted as *weakly invariant set* if all the transitions that leave the states in the subset X' are transitions with controllable events: [30]

$$\forall x \in X' \text{ and } \sigma \in \Sigma_{\mathrm{u}} \text{ it holds that } \delta(x,\sigma)! \Rightarrow \delta(x,\sigma) \in X' \tag{2.25}$$

We note that this notion of invariance for automata can be considered as a closure property and should not be confused with the notion of invariance for linear systems.

Definition 2. *Let $A \subseteq X' \subseteq X$ and consider that A, X' are invariant sets in G. A is denoted as a* strong attractor *for X' in G if*

- *the strict subautomaton of G with the state set $X' \setminus A$ is acyclic*
- *$\forall x \in X'$, there is $u \in \Sigma^\star$ s.t. $\delta(x,u) \in A$*

Briefly, Definition 2 means that there must be no arbitrarily long strings outside the state set A and the set A must be reached after a limited number of event occurrences in the system. We also need to mention that the computational complexity of verifying this condition is $\mathcal{O}(|X| + |\Sigma|)$. When we think the example automaton G with $A = \{5, 9\}$ and $X' = \{2, 3, 5, 7, 8, 9\}$ as in Figure 2.23, A is a strong attractor for X' in G since G' is acyclic.

Figure 2.24 shows another example automaton G with $A = \{5, 9\}$ and $X' = \{2, 3, 4, 5, 7, 8, 9\}$. In this case, A is not a strong attractor for X' in G because G' is not acyclic and there is no path from state 4 to A.

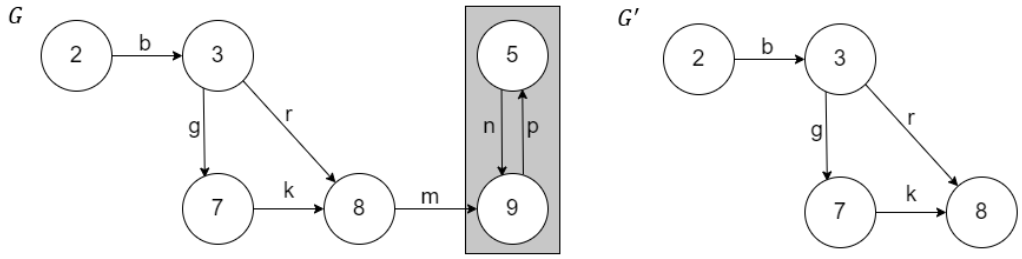

Figure 2.23: Strong attractor example.

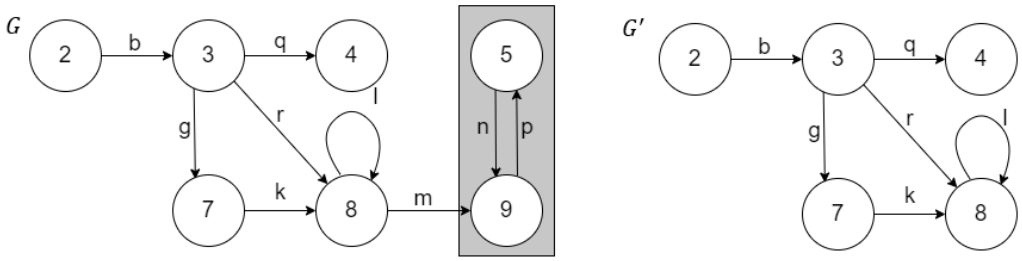

Figure 2.24: Strong attractor counterexample.

Definition 3. *Let $A \subseteq X' \subseteq X$ and consider that A, X' are invariant sets in G. $\Sigma_{\mathrm{u}} \subseteq \sigma$ be the set of uncontrollable events. A is denoted as a* weak attractor *for X' in G if there was a state feedback supervisor $S \sqsubseteq G$ s.t. A is a strong attractor for X' in S.*

In words, the set is weak attractor if there exists a supervisor that makes the closed loop system $G||S$ a strong attractor for X' in S.

When we think the example automaton G which is given in Figure 2.21 and set of uncontrollable events $\Sigma_u = \{\texttt{a}, \texttt{c}, \texttt{e}, \texttt{g}, \texttt{k}, \texttt{m}, \texttt{n}, \texttt{p}\}$, Figure 2.25 shows an example state-feedback supervisor S with $A = \{\texttt{5}, \texttt{9}\}$. In this case, A is a weak attractor for $X' = \{\texttt{3}, \texttt{5}, \texttt{7}, \texttt{8}, \texttt{9}\}$ in G using supervisor S.

Moreover, based on the references [30, 39], there exists a supremal subset of X denoted as the set $\Omega_G(A) \subseteq X$ such that A is a weak attractor for $\Omega_G(A)$ in the plant G with the uncontrollable events Σ_{u}. The computational complexity of the algorithm that computes the set $\Omega_G(A)$ is ($\mathcal{O}(|X| \cdot |\Sigma|)$). $|X|$ represents number of state

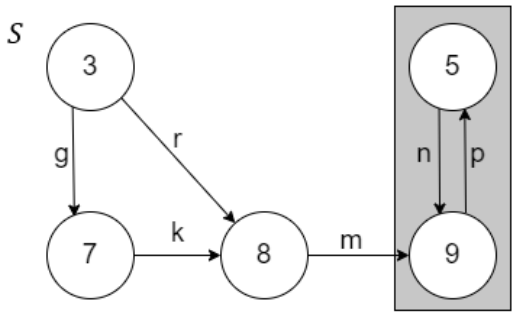

Figure 2.25: State-feedback supervisor S.

whereas, $|\Sigma|$ represents number of events used in the system, respectively. Finally, we need to mention that the computational complexity of the algorithm that obtains the supervisor ($S \sqsubseteq G$) that makes A strong attractor for $\Omega_G(A)$ in the supervisor S is $\mathcal{O}(|X|^2)$.

The literature provides polynomial-time methods for computing supervisors for state attraction [39,40]. For example, [39] determine a *minimally restrictive optimal (state-feedback) supervisor* S with complexity $\mathcal{O}(|X|^2)$.

2.9 Motivation and Problem Statement

The supervisory controller of a DES can be changed depending on the active system configuration using reconfiguration control. One of the requirements that have to be taken into account is that a suitable design strategy should be scalable to large-scale systems when designing a reconfiguration controller. This requirement is important because in large-scale systems, as the number of components of the system grows, the number of plant states grows exponentially and this is called state space explosion. Abstraction-based supervisory control can be applied to large-scale DES to handle this problem. In this thesis, we use the idea of modular control and compute suitable abstractions for each module. Then, we compute separate supervisors for the modular components and the obtained abstraction. In a reconfigurable manufacturing system, it is desired to finish one configuration's operation before moving on to the next. This enables the completion of active configuration products and it requires state attraction.

Even if we can compute supervisors for the normal operation, we cannot compute supervisors for state attraction at the current state of research since no methods are available in the existing literature. Previous work only formulates conditions but there are no analysis or synthesis algorithms.

The main subject of this thesis is filling the remaining gaps in the literature. We introduce the concept of a strong attraction-preserving abstraction and define conditions on abstractions that are suitable for state attraction. We further describe the methodology and algorithm for the computation of attraction-preserving abstractions and provide illustrations by negative and positive examples. We also formulate the concept of a composed invariant set and a composed attractor. Using this concept together with attraction-preserving abstractions, we compute supervisors that are suitable for modular and abstraction-based state attraction. These supervisors allow switching from the normal operation of the system to state attraction when performing a reconfiguration. We also clarify our methodology by an illustrative example.

CHAPTER 3

ABSTRACTION-BASED STATE ATTRACTION

The existing literature does not consider the computation of supervisor for state-attraction using the idea of abstractions. This chapter develops the new idea of abstraction-based supervisor computation for state-attraction based on the notion of attraction-preserving natural observers as introduced in Section 3.1. Furthermore, Section 3.2 develops an algorithm for the computation of attraction-preserving natural observers and Section 3.3 defines and applies the new concept of abstraction-based state-feedback supervisor.

3.1 Strong Attraction-Preserving Abstraction

In order to motivate the problem addressed in this section, we consider the example automaton G in Figure 3.1. Here, the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \alpha, \beta, \gamma\}$, the uncontrollable events are $\Sigma_{\mathrm{u}} = \{\mathtt{a}, \mathtt{b}, \mathtt{d}, \beta, \gamma\}$ and the abstraction alphabet is $\hat{\Sigma} = \{\alpha, \beta, \gamma\}$ such that $p : \Sigma^\star \to \hat{\Sigma}^\star$ is a natural observer. The equivalence classes are also shown in the figure.

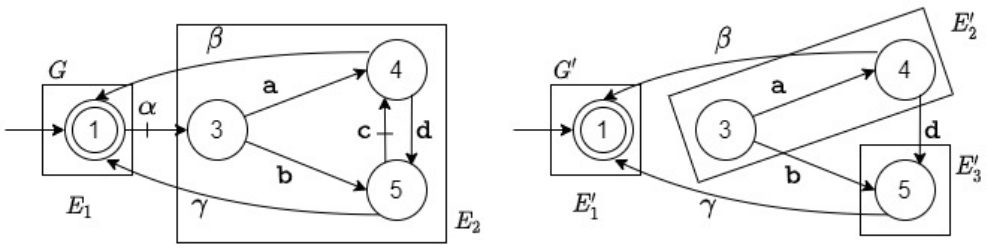

Figure 3.1: p is a natural observer for $L_{\mathrm{m}}(G)$ but not for $L_{\mathrm{m}}(G')$.

Nevertheless, when computing a state feedback supervisor G' for $A = \{1\}$ in G, as

shown in the figure, it turns out that p is no longer a natural observer for $L_{\mathrm{m}}(G')$. In particular, disabling the controllable event c in order to remove the cycle between the states 4 and 5, changes the equivalence classes as can be seen in the figure.

In order to capture this effect, we first introduce a special automaton $H_G = (X, \Sigma, \delta_G, x_0, X_{\mathrm{m}})$ from G using the equivalence classes $E \in X/\mu$ corresponding to a natural projection $p : \Sigma^\star \to \hat{\Sigma}^\star$ with the induced equivalence relation μ. Let Σ_{u} be the set of uncontrollable events and $G_E = (E, \Sigma, \delta_E, -, -)$ be the strict subautomaton of G with state set E (we are not interested in the initial state and final states of G_E). Defining $X_E = \Omega_{G_E}(E_{\mathrm{ex}})$, we further write $S_E = (X_E, \Sigma, \nu_E, -, -)$ for a state-feedback supervisor such that E_{ex} is a strong attractor for X_E in G_E.

An example for this construction is shown for the equivalence class E_1 of the example in Figure 3.1. Here, the exit states are highlighted in gray. It can be seen that the supervisor S_{E_2} disables the event c in order to remove the loop between the states 4 and 5. As a result, $E_{2,\mathrm{ex}} = \{4, 5\}$ is a strong attractor for $\Omega_{G_{E_2}}(E_{2,\mathrm{ex}}) = \{3, 4, 5\}$.

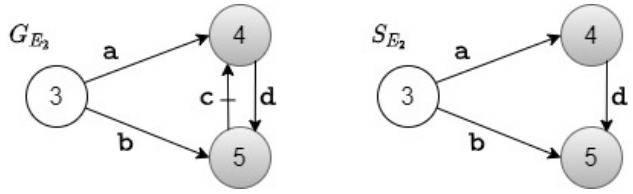

Figure 3.2: Subautomaton G_{E_2} and state-feedback supervisor S_{E_2}.

The idea for the construction of H_G is now to replace each subautomaton G_E by the corresponding state-feedback supervisor S_E in G. Since this replacement only affects transitions with events in $\Sigma \setminus \hat{\Sigma}$, all transitions with events in $\hat{\Sigma}$ are directly taken from G. Formally, this means that we define the transition relation δ_G of H_G such that

- $\forall E \in X/\mu, \forall x \in E, \forall \sigma \in (\Sigma \setminus \hat{\Sigma}) : \nu_E(x, \sigma)! \Rightarrow \delta_G(x, \sigma) = \nu_E(x, \sigma)$,
- $\forall E \in X/\mu, \forall x \in E, \forall \sigma \in \hat{\Sigma} : \delta(x, \sigma)! \Rightarrow \delta_G(x, \sigma) = \delta(x, \sigma)$.

The automaton H_G for the example in Figure 3.1 is shown in equal to the automaton G' in the same figure. After defining H_G, it is now possible to introduce the notion of an attraction-preserving natural observer as in Definition 4.

Definition 4. *Let $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$ be a plant automaton with alphabet Σ and let $p : \Sigma^\star \to \hat{\Sigma}^\star$ be a natural projection with $\hat{\Sigma} \subseteq \Sigma$ that fulfills the natural observer condition. Also assume that H_G is constructed as described above using the alphabet Σ_u of uncontrollable events. Then, p is an abstraction-preserving natural observer for G if*

1. *p is a natural observer for H_G,*
2. *$p(L_\mathrm{m}(H_G)) = p(L_\mathrm{m}(G))$.*

In words, Definition 4 states that p is an attraction-preserving natural observer for G if it is possible to apply state attraction locally (only considering events in $\Sigma \setminus \hat{\Sigma}$ within the equivalence classes induced by p) without changing 1. the validity of the natural observer property and 2. the language of the abstracted automaton. Specifically, this implies that there is an automaton $\hat{G}$ such that $L_\mathrm{m}(\hat{G}) = p(L_\mathrm{m}(G)) = p(L_\mathrm{m}(H_G))$. Since the conditions in Definition 4 are violated for the Example in Figure 3.1, we consider another example in Figure 3.3. In this example, the abstraction alphabet is $\hat{\Sigma} = \{\alpha, \beta\}$ and there are two equivalence classes induced by the projection p. When computing H_G, the same equivalence classes are obtained and both G and H_G have the same abstraction automaton $\hat{G}$.

A different situation is observed when changing the automaton G as can be seen in Figure 3.4. Here, the transition with event β is removed from state 15 compared to the previous example. Then, p is still a natural observer for $L_\mathrm{m}(G)$ but p is no longer a natural observer for $L_\mathrm{m}(H_G)$. This can be seen in Figure 3.4, where H_G now has 3 equivalence classes and there is a transition with the event $\mathtt{h} \in \Sigma \setminus \hat{\Sigma}$ between E_2 and E_3. The reason for the violation of the natural observer condition is that the transition with event $\mathtt{i}$ has been removed when computing the state-feedback supervisor for state attraction S_{E_2} in G. As a result, it is the case that there is no longer a path with only events in $\Sigma \setminus \hat{\Sigma}$ from state 15 to state 14, where the event $\beta \in \hat{\Sigma}$ is possible. It is interesting to note that the abstraction automaton for G and H_G is still the same as shown by $\hat{G}$ in the figure. Nevertheless, condition 1. in Definition 4 is violated.

After giving some intuition about attraction-preserving natural observers, we next determine a sufficient condition that allows checking if a given projection p is an

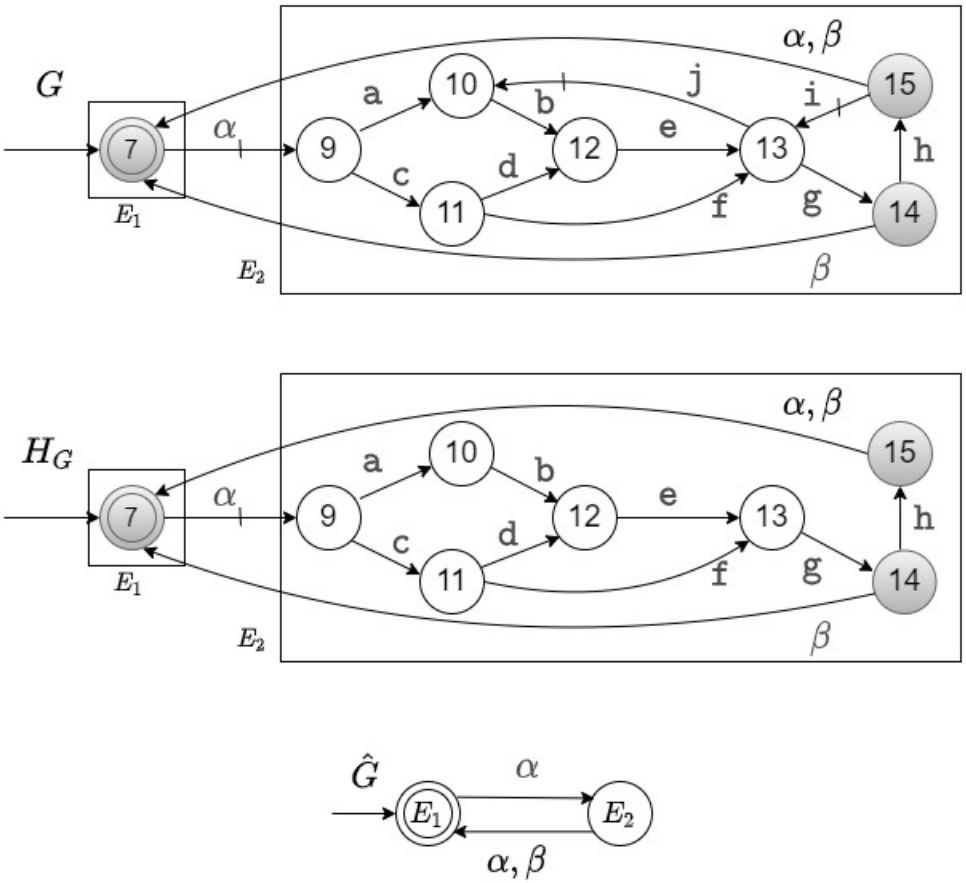

Figure 3.3: Attraction-preserving natural observer illustration: positive case.

attraction-preserving natural observer for a given plant G with the uncontrollable events Σ_{u}.

Theorem 3. *Let $G = (X, \Sigma, \delta, x_0, X_{\mathrm{m}})$ be a plant automaton with alphabet Σ and let $p : \Sigma^\star \to \hat{\Sigma}^\star$ be a natural projection with $\hat{\Sigma} \subseteq \Sigma$ that fulfills the natural observer condition. Also assume that μ is the equivalence relation on X defined by p with the corresponding set of equivalence classes X/μ and the canonical projection cp_μ. In addition, for each equivalence class $E \in X/\mu$, we write $E_{\mathrm{ex}} \subseteq E$ for the exit states in E. Finally, for each $x \in X$, we write $\hat{\Sigma}(x) = \{\sigma \in \hat{\Sigma} | \delta(x, \sigma)!\}$ for the high-level events possible at state x. Then, the abstraction p is an attraction-preserving natural observer for G if it holds that*

1. *for each $E \in X/\mu$, $\Omega_{G_E}(E_{\mathrm{ex}}) = E$,*
2. *for all $E \in X/\mu$ and $\forall x, x' \in E_{\mathrm{ex}}$, $\hat{\Sigma}(x) = \hat{\Sigma}(x')$ or $\hat{\Sigma}(x) \subset \hat{\Sigma}(x')$ and*

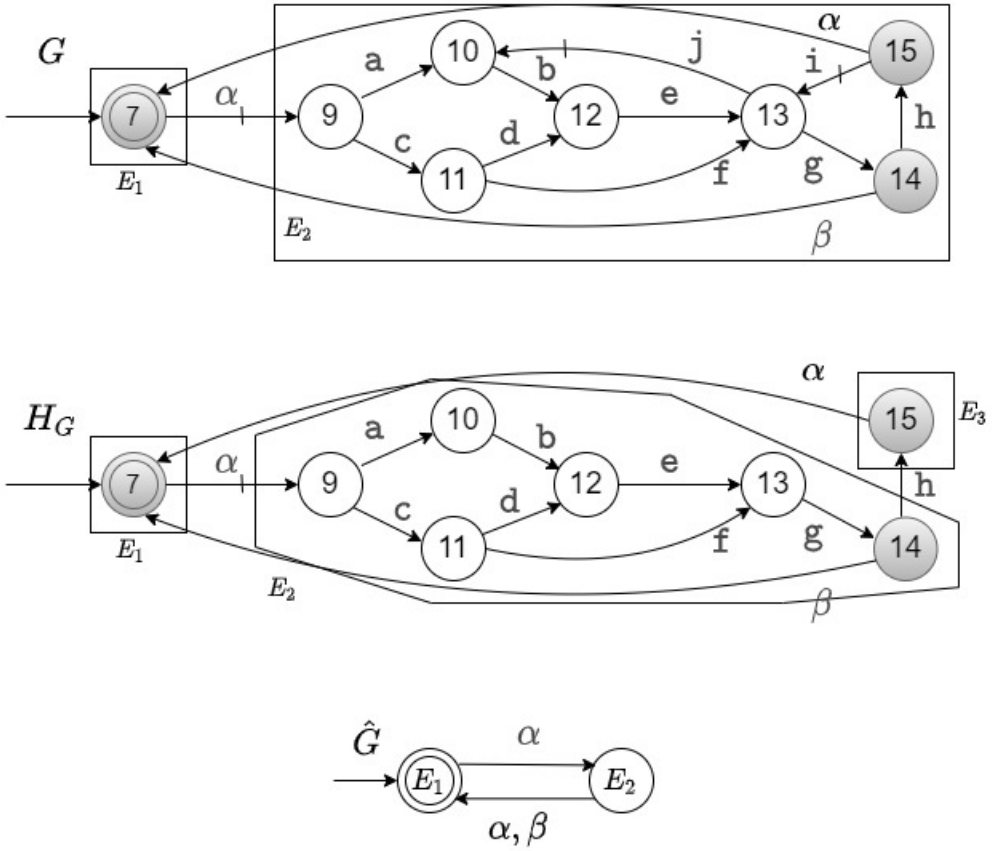

Figure 3.4: Attraction-preserving natural observer illustration: negative case.

$$x \in \Omega_{G_E}(\{x'\})$$

In words, the first condition states that the optimal set of weak attraction for each subautomaton G_E and the set of exit state E_{ex} must be equal to E. That is, when computing a state-feedback supervisor for state attraction with the plant automaton G_E and the target set E_{ex}, it should be possible to reach E_{ex} from any state in E within a bounded number of event occurrences. The second condition is based on the fact that p is a natural observer. That is, p defines an equivalence relation μ on the states X of G. In addition, we know from Section 2.6.2 that there are no bad exit state pairs (according to the natural observer condition) in any equivalence class $E \in X/\mu$. However, when computing the supervisor S_E for state attraction when constructing H_G, some of the transitions will be disabled such that potentially additional bad state pairs can be created. Then, it is ensured that there are no additional bad state pairs

when applying state attraction if condition 2. is fulfilled. That is, for each equivalence class E, either all exit states have the same outgoing high-level transitions or the set of outgoing high-level transition of an exit state x is a subset of the outgoing high-level transitions of another exit state x' and x' can be reached from x even when applying state attraction.

This condition is fulfilled in the example in Figure 3.3. Here, there are two equivalence classes E_1 and E_2. E_1 only has a single state such that condition 1. and 2. in Theorem 3 are trivially fulfilled. Looking at E_2, the set of exit states is $E_{2,\mathrm{ex}} = \{14, 15\}$ and $\Omega_{G_{E_2}}(E_{2,\mathrm{ex}}) = \{9, 10, 11, 12, 13, 14, 15\}$. That is condition 1. is fulfilled. In addition, it holds that $\hat{\Sigma}(14) = \{\beta\} \subset \hat{\Sigma}(15) = \{\alpha, \beta\}$. Since $14 \in \Omega_{G_{E_2}}(\{15\})$, condition 2. is also fulfilled. That is, p is an attraction-preserving natural observer for this example.

We next provide a proof of Theorem 3.

Proof. We assume that condition 1. and 2. in the theorem are fulfilled and want to show that

1. p is a natural observer for H_G,
2. $p(L_\mathrm{m}(H_G)) = p(L_\mathrm{m}(G))$.

in line with Definition 4.

Since p is a natural observer by assumption, it is sufficient to show condition 2. that is, we pick an arbitrary string $t \in p(L(G))$ and have to show that $t \in p(L(H_G))$. Since $t \in p(L(G))$, there must be a $s \in L(G)$ such that $p(s) = t$ and hence $\delta(x_0, s)!$. Furthermore, we know that $x_0 \in E$ for some equivalence class $E \in X/\mu$. Writing $\hat{G} = (\hat{X}, \hat{\Sigma}, \hat{\delta}, \hat{x}_0, \hat{X}_\mathrm{m})$ for the abstraction automaton of G, we also know that $\hat{x}_0 = cp_\mu(x_0)$ is the corresponding state of x_0 in $\hat{G}$. Now we can write $t = \sigma_1\sigma_2\cdots\sigma_m$, with events $\sigma_i \in \hat{\Sigma}$ for $i = 1, \ldots, m$. Since $p(s) = t \in p(L(G))$, it further holds that $\hat{\delta}(\hat{x}_0, \sigma_1)!$. That is, there must be a state $x' \in E$ such that $\delta(x', \sigma_1)!$. Since $\sigma_1 \in \hat{\Sigma}$ is an event in the abstraction alphabet, this also implies that x' must be an exit state in E_ex. Moreover, because of condition 1. in the theorem, there must be some exit state $x'' \in E_\mathrm{ex}$ and a $u'' \in (\Sigma \setminus \hat{\Sigma})^\star$ such that $\delta_G(x_0, u'') = x''$. If the event σ_1 is not

defined at x'', then there must be a string $u' \in (\Sigma \setminus \hat{\Sigma})^\star$ to a state $x' \in E_{\text{ex}}$ where σ_1 is defined because of condition 2. in the theorem. That is, writing $u_1 = u''u'$, $\delta_G(x_0, u_1\sigma_1)!$ and leads to the equivalence class that corresponds to $\hat{\delta}(\hat{x}, \sigma_1)$. Then, the same argument can be repeated for the remaining events $\sigma_2, \ldots, \sigma_m$. That is together, there is a string $u = u_1\sigma_1 u_2\sigma_2 \cdots u_m\sigma_m \in \Sigma^\star$ such that $p(u) = t$ and $\delta_G(x_0, u)!$. Thus, $u \in L(H_G)$ and $p(u) = t \in p(L(H_G))$. Since the string t was arbitrary, this concludes the proof. □

We next provide additional examples in order to further illustrate Theorem 3. Figure 3.5 shows an example plant automaton G_1 with alphabet $\Sigma_1 = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}\}$. Let $p_1 : \Sigma_1^\star \to \hat{\Sigma}_1^\star$ be a natural projection with abstraction alphabet $\hat{\Sigma}_1 = \{\texttt{a}, \texttt{b}, \texttt{d}, \texttt{e}\}$. It can be verified that p_1 is a natural observer for $L_{\text{m}}(G_1)$ and Figure 3.6 shows the equivalence classes defined by p_1 as well as the abstracted automaton $\hat{G}_1$.

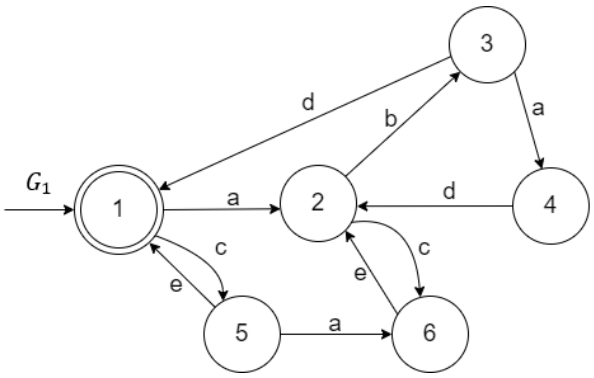

Figure 3.5: Example automaton G_1 for strong attraction-preserving abstraction.

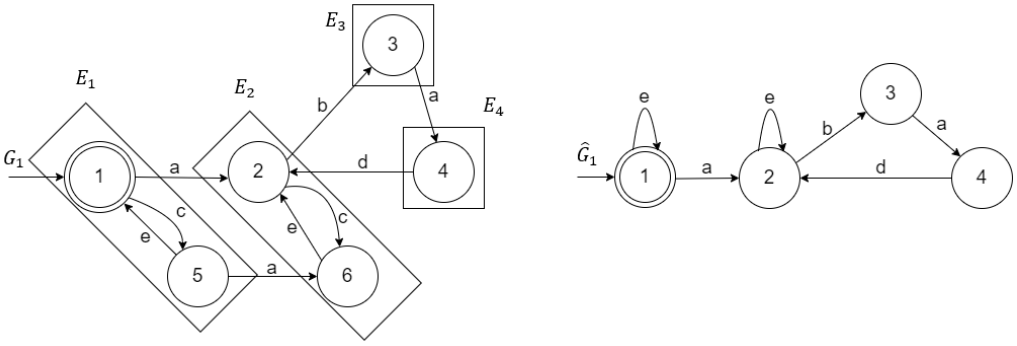

Figure 3.6: Equivalence classes defined by p_1 and abstraction of G_1.

Regarding condition 1. in Theorem 3, we can see that E_3 and E_4 only have a single

state. Moreover, G_{E_1} and G_{E_2} are shown in Figure 3.7. It can be seen that both of them contain all states of the respective equivalence class. That is, condition 1. of the theorem is fulfilled.

Figure 3.7: Subautomata G_{E_1} and G_{E_2}.

Regarding condition 2. in Theorem 3, there are four equivalence classes defined by p_1. For E_1, the set of exit states is $E_{\text{ex1}} = \{1, 5\}$. It holds that $\hat{\Sigma}(1) = \{\mathtt{a}, \mathtt{c}\}$ and $\hat{\Sigma}(5) = \{\mathtt{a}\}$, that is, $\hat{\Sigma}_1(5) \subset \hat{\Sigma}_1(1)$. In addition, it can be observed from Figure 3.7 that $5 \in \Omega_{G_1}(\{1\})$. Hence, this condition is also fulfilled. For E_2, E_3 and E_4 set of exit states are $E_{\text{ex2}} = \{2\}$, $E_{\text{ex3}} = \{3\}$ and $E_{\text{ex4}} = \{4\}$ respectively and since there is only one exit state in each set of exit states, both conditions are fulfilled. Together, condition 2. in the theorem is fulfilled for all equivalence classes. Hence, the abstraction p_1 is a strongly attraction-preserving natural observer for $L_{\text{m}}(G_1)$ and A.

We can further inspect the plant automaton G_2 in Figure 3.8 as another example and let p_2 be an abstraction with abstraction alphabet $\hat{\Sigma}_2 = \{\alpha, \beta\}$. In this example, there is only one equivalence class which contains all states of G_2, in other words $E = \{1, 2, 3, 4, 5\}$ and $E_{\text{ex}} = \{3, 5\}$. Since $\hat{\Sigma}(3) = \{\alpha\}$ and $\hat{\Sigma}(5) = \{\beta\}$, the second condition in Theorem 4 is not fulfilled thus, the abstraction p_2 is not a strongly attraction-preserving natural observer for G_2 and any $A \subseteq X_2$. Figure 3.9 shows the abstraction automaton $\hat{G}_2$ as well as the automaton H_{G_2} with its equivalence classes. It can be seen that there are transitions with event $\mathtt{g} \in \Sigma_2 \backslash \hat{\Sigma}_2$ between the equivalence classes $E_1 = \{1, 2, 3\}$ and $E_2 = \{4, 5\}$, which also clarifies the violation of Theorem 3.

Figure 3.10 shows one more example automaton G_3 and let p_3 be an abstraction with abstraction alphabet $\hat{\Sigma}_3 = \{\alpha, \beta\}$. In this example, there is only one equivalence class with $E = \{1, 2, 3, 4, 5\}$ and $E_{\text{ex}} = \{3, 5\}$. The set of outgoing high-level transitions of the exit states are $\hat{\Sigma}(3) = \{\alpha, \beta\}$ and $\hat{\Sigma}(5) = \{\beta\}$. Since $\hat{\Sigma}(5) \subset \hat{\Sigma}(3)$ and $5 \in \Omega_E(\{3\})$ the abstraction p_3 is a strongly attraction-preserving for G_3. Figure

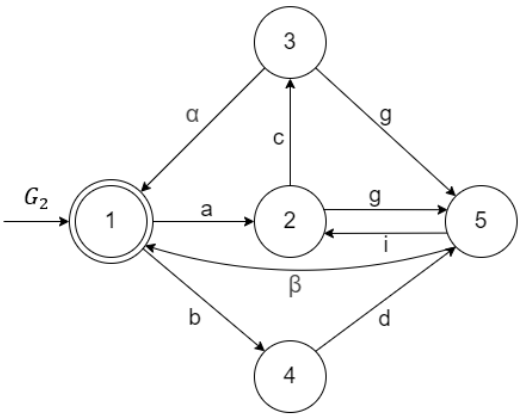

Figure 3.8: Example automaton G_2 for strong attraction-preserving abstraction.

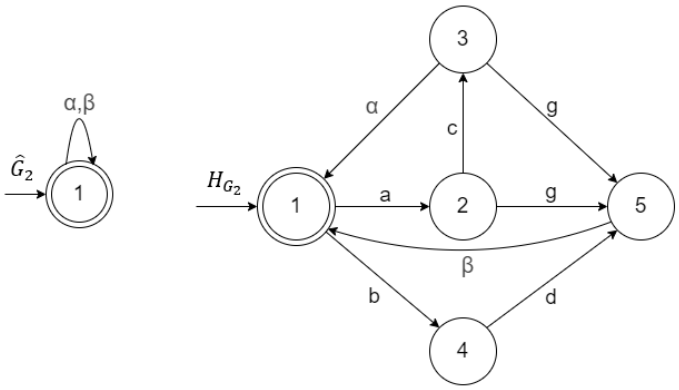

Figure 3.9: $\hat{G}_2$ and H_{G_2}.

3.11 shows abstraction automaton $\hat{G}_3$ as well as the automaton H_{G_3}.

Figure 3.12 shows a more complex example automaton G_4 and let p_4 be an abstraction with abstraction alphabet $\hat{\Sigma}_4 = \{\alpha, \beta, \gamma\}$. Figure 3.13 shows the three equivalence classes defined by p_4. If we look at the equivalence class E_3, the set of exit states is $E_{4,\text{ex}} = \{17, 18\}$. Since $\hat{\Sigma}_4(17) = \{\alpha\}$ and $\hat{\Sigma}_4(18) = \{\beta\}$, condition 2. in Theorem 3 is not fulfilled. Thus, p_4 is not a strongly attraction-preserving abstraction for G_4.

If we add $a1, b1, s1, p1$ to the abstraction alphabet $\hat{\Sigma}_4$, we get the equivalence classes shown in Figure 3.14 for the same example. In that case, both conditions in Theorem 3 are fulfilled and p_4 is a strongly attraction-preserving abstraction for G_4. The resulting abstracted automaton $\hat{G}_4$ is shown in Figure 3.15.

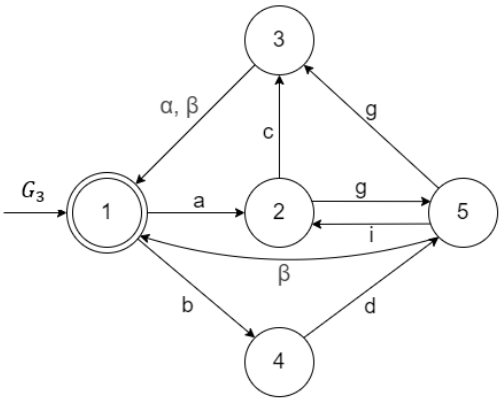

Figure 3.10: Example automaton G_3 for strong attraction-preserving abstraction.

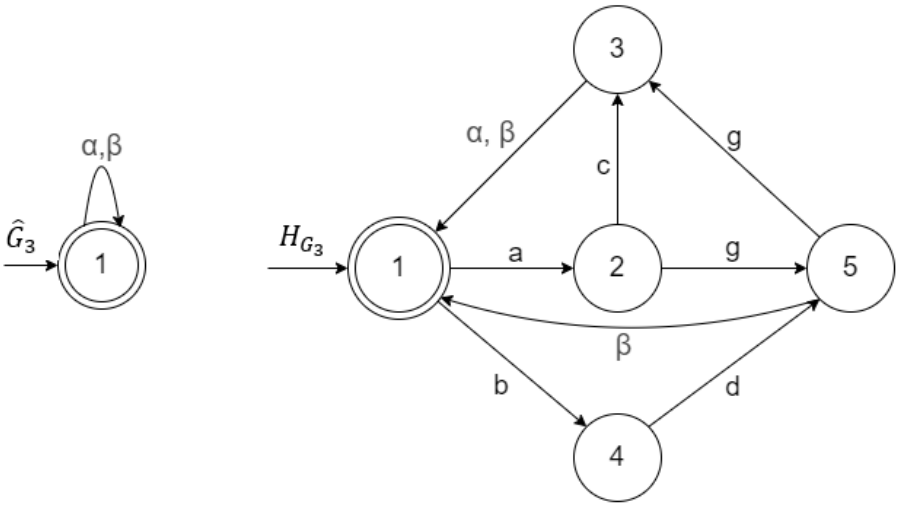

Figure 3.11: $\hat{G}_3$ and H_{G_3}.

3.2 Computation of Attraction-Preserving Abstraction

When looking at G and H_G in Figure 3.4, it can be seen that there is a way to achieve an attraction-preserving natural observer by adding the high-level events `h` and `i`. Here, the important observation is that this works because we want to put the states 14 and 15 in different equivalence classes. Specifically, these two states violate condition 2. in Theorem 3 because $\hat{\Sigma}(15) = \{\alpha\}$ and $\hat{\Sigma}(14) = \{\beta\}$. That is neither $\hat{\Sigma}(15) \subseteq \hat{\Sigma}(14)$ nor $\hat{\Sigma}(14) \subseteq \hat{\Sigma}(15)$. That is, in line with the discussion in Section 2.6.3, the pair $(14, 15)$ should be identified as a bad exit state pair, which has to be split by adding events to the high-level alphabet. In the example, the two events `i` and `h` are

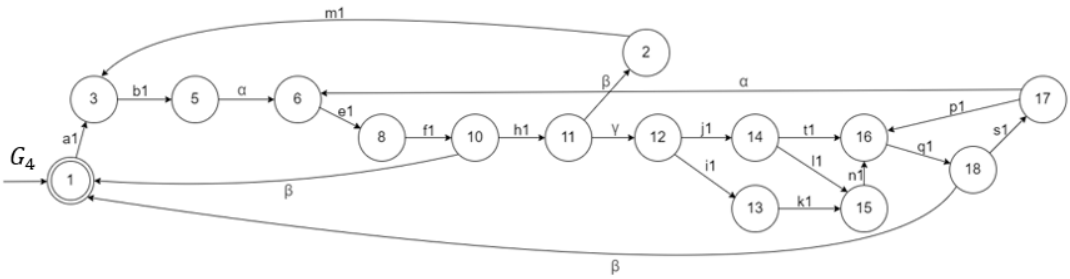

Figure 3.12: Example automaton G_4 for strong attraction-preserving abstraction.

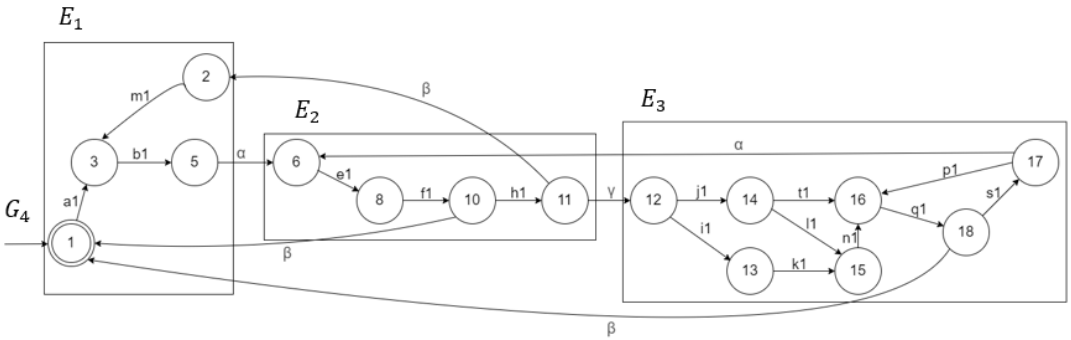

Figure 3.13: Equivalence classes defined by p_4.

suitable for this purpose. The resulting automata G and H_G with their equivalence classes are shown in Figure 3.16.

In general, similar to the discussion in Section 2.6.3, we again try to identify bad exit state pairs. Specifically, we look at the sets of abstraction events that are possible at different exit states. Consider two generic exit states x_1 and x_2 in the same equivalence class E and denote their outgoing abstraction events as $\Sigma_1 = \hat{\Sigma}(x_1)$ and $\Sigma_2 = \hat{\Sigma}(x_2)$. Then, we distinguish the following cases:

1. $\hat{\Sigma}_1 = \hat{\Sigma}_2$: in this case (x_1, x_2) is not a bad state pair,

2. $\hat{\Sigma}_1 \subset \hat{\Sigma}_2$ in this case, (x_1, x_2) is not a bad state pair if $x_1 \in \Omega_{G_E}(\{x_2\})$. That is, when computing an attractor, x_2 is reachable from x_1, which implies that the same abstraction events are possible from x_1 and x_2. If the condition is violated, (x_1, x_2) is a bad exit state pair,

3. $\hat{\Sigma}_2 \subset \hat{\Sigma}_1$: This is the reverse case of item 2.,

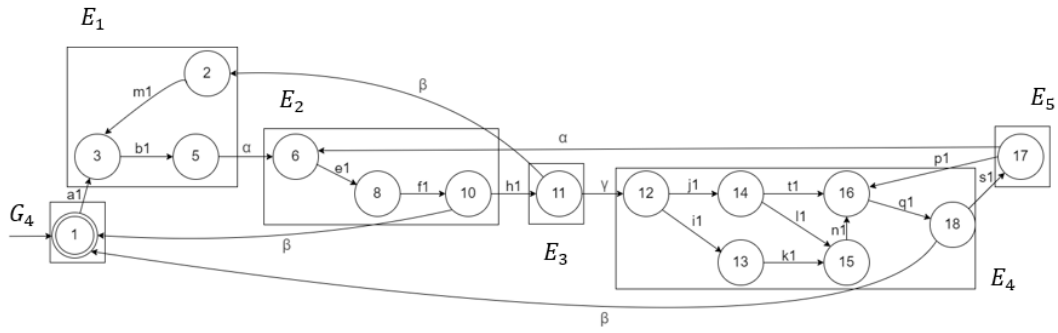

Figure 3.14: Equivalence classes defined by new abstraction p_4.

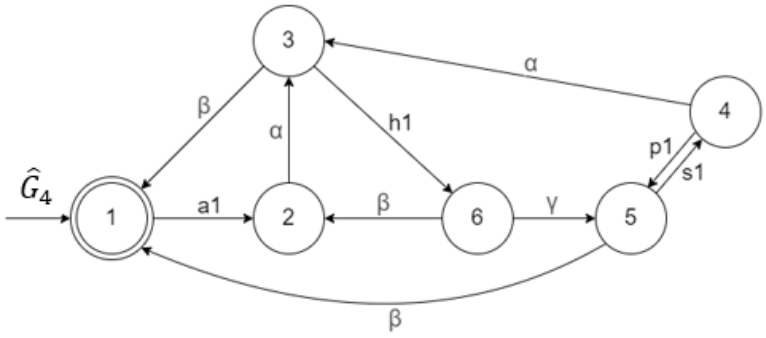

Figure 3.15: Abstracted automaton $\hat{G}_4$ for strong attraction-preserving abstraction.

4. $\hat{\Sigma}_1 \setminus \hat{\Sigma}_2 \neq \emptyset \wedge \hat{\Sigma}_2 \setminus \hat{\Sigma}_1 \neq \emptyset$:: In this case, (x_1, x_2) is a bad exit state pair since different abstraction events are possible from these exit states. Hence, x_1 and x_2 should not be in the same equivalence class.

After finding the bad exit state pairs as described above, we can apply the same algorithm for splitting bad exit state pairs as presented in Section 2.6.3. Hereby, it is again guaranteed that the algorithm terminates since only a finite number of events can be successively added to the abstraction alphabet. Specifically, it is the case that $\hat{\Sigma} = \Sigma$ in the worst case. If $\hat{\Sigma} = \Sigma$, the conditions in Definition 4 are trivially fulfilled. The described algorithm was implemented in the open-source C++ library libfaudes for DES [41].

Figure 3.17 shows an example automaton G_2 with two different abstraction alphabets. For abstraction alphabet $\hat{\Sigma}_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$, there are three equivalence classes E_1, E_2 and E_3 defined by this abstraction. If we consider E_3, the set of exit states is

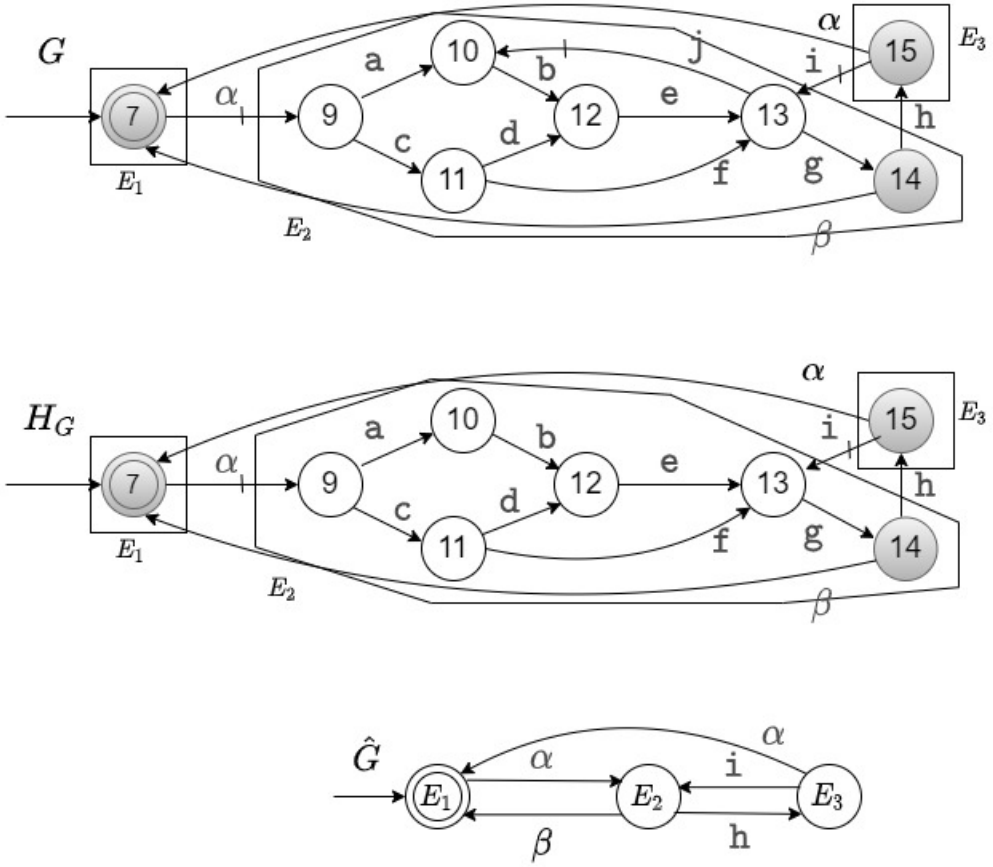

Figure 3.16: Attraction-preserving natural observer illustration: Modified alphabet $\hat{\Sigma}$.

$E_{\text{ex}} = \{3, 5\}$. Since $\hat{\Sigma}(3) = \{\mathtt{e}\}$ and $\hat{\Sigma}(5) = \{\mathtt{g}\}$, it is not a strongly attraction-preserving abstraction for G_2 because $(3, 5)$ is a bad state pair. In order to overcome this problem, we can add event d to the abstraction alphabet. The new abstraction alphabet $\hat{\Sigma}_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$ and equivalence classes E_1, E_2, E_3 and E_4 can be seen in Figure 3.17 (right). In this case, since state 3 and 5 are no longer in the same equivalence class, there is no more bad state pair. That is, p_2 is an attraction-preserving natural observer for G_2.

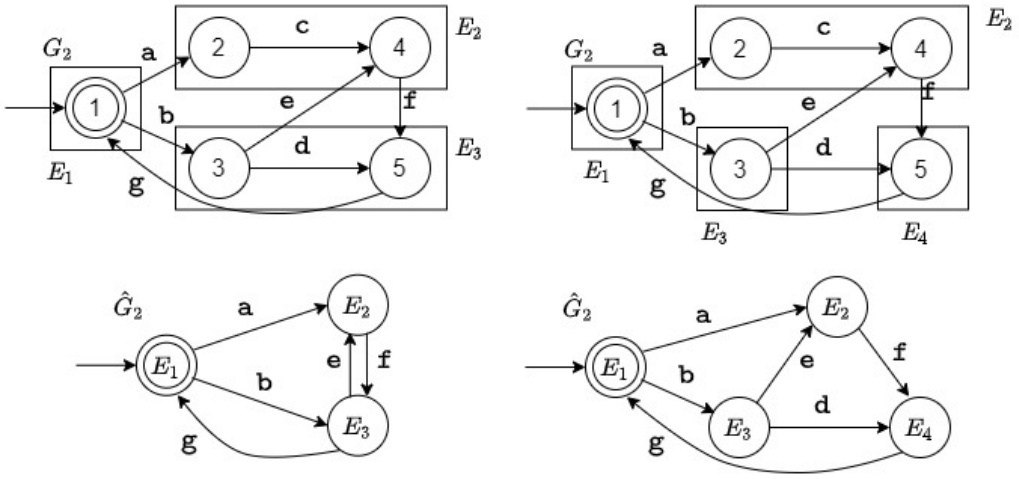

Figure 3.17: Equivalence classes for different abstraction alphabets for G_2: $\hat{\Sigma} = \{\mathtt{a}, \mathtt{b}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$ (left); $\hat{\Sigma} = \{\mathtt{a}, \mathtt{b}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$ (right).

3.3 Abstraction-based State-Feedback Supervisor

In the previous sections, we defined the notion of an attraction-preserving natural observer and presented sufficient conditions for checking this condition. In addition, we were able to develop an algorithm for extending the abstraction alphabet in order to obtain an attraction-preserving natural observer based on the definition of bad exit state pairs. This section highlights the actual benefit of using attraction-preserving natural observers. To this end, we first define an operation that characterizes the application of a supervisor for state attraction on the abstracted level to the original plant.

Definition 5. *Let $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$ be a plant automaton with alphabet Σ and uncontrollable events Σ_u. Further, let $p : \Sigma^\star \to \hat{\Sigma}^\star$ be a natural projection with $\hat{\Sigma} \subseteq \Sigma$ and assume that p is a natural observer with the abstracted plant automaton $\hat{G} = (\hat{X}, \hat{\Sigma}, \hat{\delta}, \hat{x}_0, \hat{X}_\mathrm{m})$. Write μ for the equivalence relation induced by μ and $cp_\mu : X \to \hat{X}$ for the corresponding canonical projection.*

Then, we define the automaton $S = (Q, \Sigma, \nu, q_0, Q_\mathrm{m}) = \hat{S} \lceil\rceil G$ for $\hat{S}$ and G as follows:

1. *$q_0 = x_0$ and $Q_\mathrm{m} = X_\mathrm{m}$,*

2. $\forall x \in X \wedge \sigma \in (\Sigma \setminus \hat{\Sigma})$: $\delta(x,\sigma)! \Rightarrow \nu(x,\sigma) = \delta(x,\sigma)$,

3. $\forall x \in X \wedge \sigma \in \hat{\Sigma}$: $\hat{\nu}(cp_\mu(x),\sigma)! \wedge \delta(x,\sigma)! \rightarrow \nu(x,\sigma) = \delta(x,\sigma)$.

We call the automaton S the abstraction-based state-feedback supervisor candidate for G.

Definition 5 assumes that a state-feedback supervisor $\hat{S}$ is computed for an abstracted plant $\hat{G}$, that is, $\hat{S} \sqsubseteq \hat{G}$. Then, the definition states how $\hat{S}$ should be applied to the original plant G. Specifically, it is considered that each state $q \in \hat{Q}$ of $\hat{S}$ is actually also a state $q \in \hat{X}$ of $\hat{G}$ since $\hat{S}$ is a subautomaton of $\hat{G}$. That is, each state $q \in \hat{Q}$ of $\hat{S}$ is associated with the corresponding states in G, which are exactly the states that belong to the equivalence class q, that is, the states $x \in X$ such that $cp_\mu(x) = q$. For all these states, all events in $\Sigma \setminus \hat{\Sigma}$ are enabled (they occur as in G) according to condition 2. in Definition 5. In addition, events in $\hat{\Sigma}$ are only enabled if they are enabled by $\hat{S}$ as is stated in condition 3. of Definition 5.

In order to illustrate Definition 5, we consider the example in Figure 3.18. Here, G constitutes the plant with its equivalence classes for the projection with alphabet $\hat{\Sigma} = \{\mathtt{a},\mathtt{b},\mathtt{d},\mathtt{e},\mathtt{g}\}$ and $\hat{G}$ is the corresponding abstracted automaton. Then, $\hat{S}$ represents a state-feedback supervisor for $\hat{G}$. Hereby, the arrows in the figure show the relation between the states of $\hat{S}$ and the corresponding equivalence classes in G. That is, the enabled at the respective states should be enabled for the corresponding equivalence classes according to Definition 5. The resulting state-feedback supervisor $S = \hat{S} \lceil\rceil G$ for G is also shown in the figure.

Using the supervisor implementation in Definition 5, Theorem 4 shows that it is possible to compute state-feedback supervisors for state attraction based on the abstracted automaton if the projection p is an attraction-preserving natural observer.

Theorem 4. *Let G, p, $\hat{\Sigma} \subseteq \Sigma$, $\hat{G}$, H_G, μ and cp_μ be defined as above. In addition, for each equivalence class $E \in X/\mu$, we write $E_{\mathrm{ex}} \subseteq E$ for the exit states in E. Assume that p is an attraction-preserving natural observer for $L_{\mathrm{m}}(G)$ and let $E_{\mathrm{ex}} \subseteq A \subseteq X$ for some $E \in X/\mu$. Suppose that $\hat{S} = (\hat{Q},\hat{\Sigma},\hat{\nu},-,-) \sqsubseteq \hat{G}$ is a state-feedback supervisor such that $cp_\mu(A)$ is a strong attractor in $\hat{Q}$ for $\hat{S}$ and compute $S = (Q,\Sigma,\nu,-,-) = \hat{S} \lceil\rceil H_G$. Then, A is a strong attractor for Q in S.*

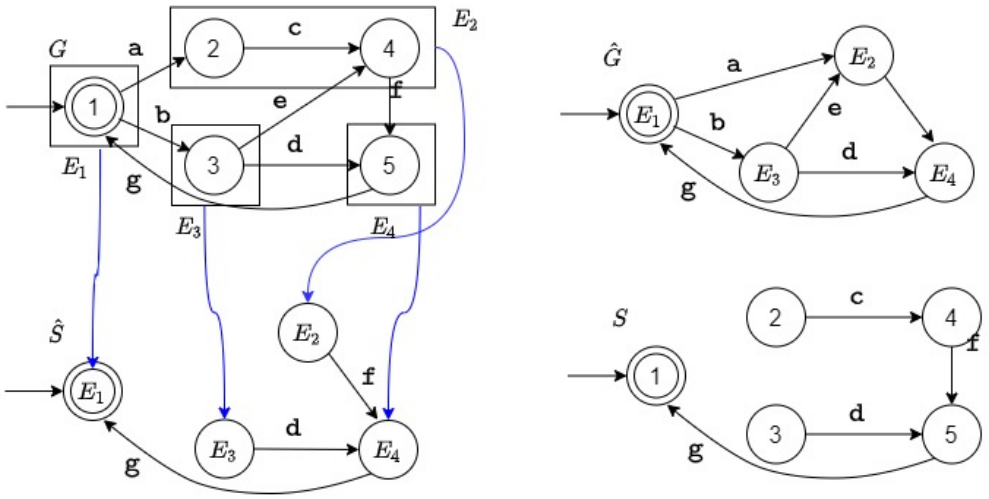

Figure 3.18: Illustration of Definition 5.

The theorem uses the operation $\lceil\rceil$ in Definition 5 to obtain a state-feedback supervisor for state attraction for the original plant G from a state-feedback supervisor for state attraction for the abstracted plant $\hat{G}$. Here, several conditions have to be fulfilled. First, it is required that the natural projection is an attraction-preserving natural observer. Second, the state-feedback supervisor $\hat{S}$ for the abstracted plant $\hat{G}$ has to be computed for a target set that is the canonical projection of the target set A of the original plant. Third, the target set for state attraction A has to include the set of exit states E_{ex} for at least one equivalence class E. This is necessary since the supervisor $\hat{S}$ only disables events in $\hat{\Sigma}$, whereas events in $\Sigma \setminus \hat{\Sigma}$ occur according to the dynamics of the automaton H_G. Looking at the definition of H_G in Section 3.1, this means that, after entering an equivalence class E, it is ensured that states in E_{ex} will be reached in a finite number of transitions. This is exactly what is required in order to fulfill the conditions Definition 2 for strong attraction. We next formally prove Theorem 4.

Proof. We need to show that A is a strong attractor for Q in S and recall that $H_G = (X, \Sigma, \delta_G, x_0, X_{\text{m}})$. Consider any state $q \in Q$. Then, by definition of the operation $\lceil\rceil$, it holds that $Q \subseteq X$. That is, $q \in X$. Furthermore, since p is an attraction-preserving natural observer, q belongs to some equivalence class E and $cp_\mu(q) = \hat{q}$ for some state $\hat{q} \in \hat{Q}$. Since $\hat{A} = cp_\mu(A)$ is a strong attractor in $\hat{Q}$ for $\hat{S}$, there must be a string $t \in \hat{\Sigma}^\star$ such that $\hat{\nu}(\hat{q}, t) \in \hat{A}$ and $|t| \leq \hat{N}$ for some $\hat{N} \in \mathbb{N}$. Then, with the

same argument as in the proof of Theorem 3, there must be a string $u \in \Sigma^\star$ such that $p(u) = t$ and $\delta_G(q, u)!$ and $|u| \leq N$ for some $N \in \mathbb{N}$. Specifically, we can write $x = \delta_G(q, u)$ and it must be the case that $cp_\mu(x) \in \hat{A}$. Accordingly, x must belong to some equivalence class E such that $E_{\text{ex}} \subseteq A$. Since $\Omega_{G_E}(E_{\text{ex}}) = E$ since p is an attraction-preserving natural observer, there must be a string $u' \in (\Sigma \setminus \hat{\Sigma})^\star$ and a state $x' \in E_{\text{ex}}$ such that $\delta_G(x, u') = x'$ and $|u'| \leq N'$ for some $N' \in \mathbb{N}$. Together, this implies that $\delta_G(q, uu') \in A$ and $uu' \leq N + N'$. Since q was chosen arbitrarily, A is a strong attractor for Q in S. □

We next apply Theorem 4 to the example in Figure 3.3. Here, $\hat{S}$ is a state-feedback supervisor for $\hat{G}$ and the target set $\{E_1\}$. Applying $\hat{S}$ to H_G in the form $\hat{S} \lceil \rceil H_G$ leads to the automaton S in the Figure 3.19. It can be seen that the state set $\{7\}$ is a strong attractor in S.

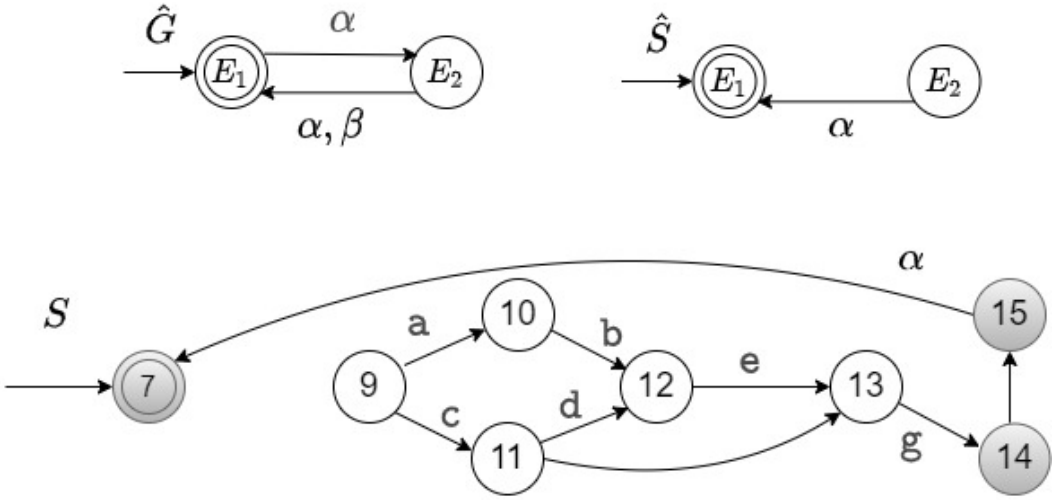

Figure 3.19: Illustration of Theorem 4, example 1.

Finally, Figure 3.20 shows the application of Theorem 4 to the example in Figure 3.16. It is again the case that the resulting supervisor S has $\{7\}$ as a strong attractor after applying the supervisor $\hat{S}$ that is computed for the abstracted plant $\hat{G}$.

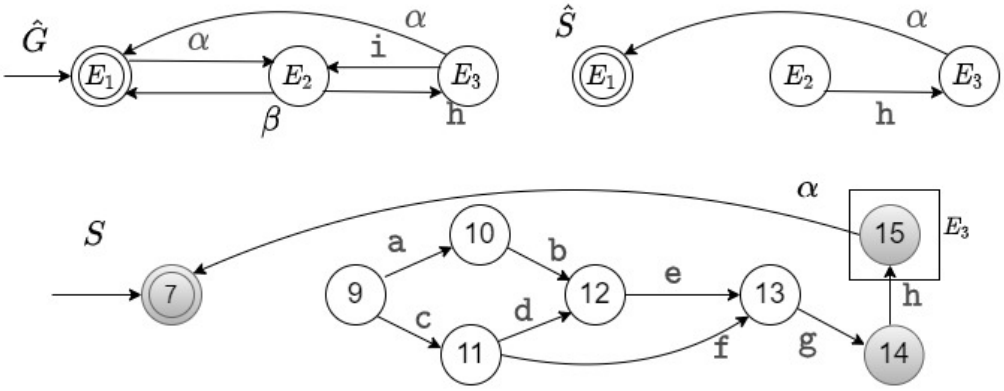

Figure 3.20: Illustration of Theorem 4, example 2.

CHAPTER 4

ABSTRACTION-BASED SUPERVISOR COMPUTATION FOR STATE ATTRACTION

Chapter 3 develops a new method for state attraction based on state-feedback supervisor that is computed for the abstracted plant. This idea helps reducing the computational effort for computing the state-feedback supervisor for state attraction since the abstracted plant usually has a smaller state space than the original plant due to the attraction-preserving natural observer condition. However, similar to the classical monolithic supervisory control theory described in Section 2.4, it is still required to compute the overall plant G in order to obtain the abstraction $\hat{G}$. As discussed in Section 2.7, this is infeasible for systems of practical size because of the well-known state space explosion problem.

As a remedy, the classical supervisory control theory introduces the idea of modular and abstraction-based supervisory control as described in Section 2.7. This chapter extends the existing ideas to the case of state attraction. In this context, it has to be noted that the basic definitions were already introduced in [28] but without computational procedures for the analysis and synthesis of supervisors. These procedures are developed in this chapter.

The organization of the chapter is as follows. Section 4.1 and 4.2 introduce the necessary definitions of a composed invariant set and a strong composed attractor. The supervisor computation method is developed in Section 4.3 and an illustrative example is presented in Section 4.4.

4.1 Composed Invariant Set

The notion of an invariant set is introduced for a single automaton in Section 2.8. We next present an extension of this condition to the case of multiple automata as introduced in [28]. The notion of a composed invariant set is defined in Definition 6.

Definition 6. *Let $T_i = (W_i, \Sigma_i, \omega_i, -, -)$ be automata for $i = 1, \ldots, l$ and let $C \subseteq W_1 \times \cdots \times W_l$ be a set of state tuples with one state from each automaton T_i. We write $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ and use the natural projections $p_i : \Sigma^\star \to \Sigma_i^\star$. C is denoted as a* composed invariant set *for $T_1, \ldots, T_l$ if it holds for all $c = (c_1, \ldots, c_l) \in C$ and $\sigma \in \Sigma$ that*

$$\begin{aligned} &\omega_i(c_i, p_i(\sigma))! \text{ for all } i = 1, \ldots, l \\ &\quad \Rightarrow (\omega_1(c_1, p_1(\sigma)), \ldots, \omega_l(c_l, p_l(\sigma))) \in C. \end{aligned} \tag{4.1}$$

The idea of the definition is as follows. We consider states that can be reached when applying the synchronous composition to the automata $T_1, \ldots, T_l$. Each such state is described by a state tuple $c = (c_1, \ldots, c_l)$. Then the definition considers a subset $C \subseteq W_1 \times \cdots \times W_l$ of all states that belong to the synchronous composition. C is called a composed invariant set if it holds for all states $c \in C$ that starting from c and following the synchronous composition of $T_1, \ldots, T_l$ as stated in (4.1), all states reached from c are again in C.

Definition 6 is illustrated by the example in Figure 4.1 with three automata T_1, T_2, T_3. The set $C = \{$`(1,1,1)`, `(2,1,2)`, `(3,1,3)`, `(4,1,4)`, `(2,2,5)`, `(2,3,5)`, `(2,4,5)`, `(6,1,2)`, `(1,2,6)`, `(1,3,6)`, `(1,4,6)`, `(5,1,1)`$\}$ consists of states that can be reached in the synchronous composition $T_1||T_2||T_3$. Then, it can be checked if C is a composed invariant set for T_1, T_2 and T_3 by looking at each state tuple of C and verifying the condition in (4.1). Consider for example state $c = (2, 1, 2) \in C$. Here, the event `b` is possible in T_1 and T_3, whereas `b` does not belong to the alphabet of T_2. Hence, there is a transition with `b` in the synchronous composition $T_1||T_2||T_3$ leading to the state $(3, 1, 3)$, which also belongs to C. In addition, the event `c` is possible in T_1 but is not possible in T_2 and T_3, which means that there is no transition with `c` from $(2, 1, 2)$. The same is true for the event `d`. Together, we verified that the only state reachable from $(2, 1, 2)$ is the state $(3, 1, 3)$, which also

belongs to C. Performing the same check for all the states in C, it can be verified that c is a composed attractor. In the scope of this thesis, the composed invariant set verification was implemented as a function in the C++ software library libfaudes [41].

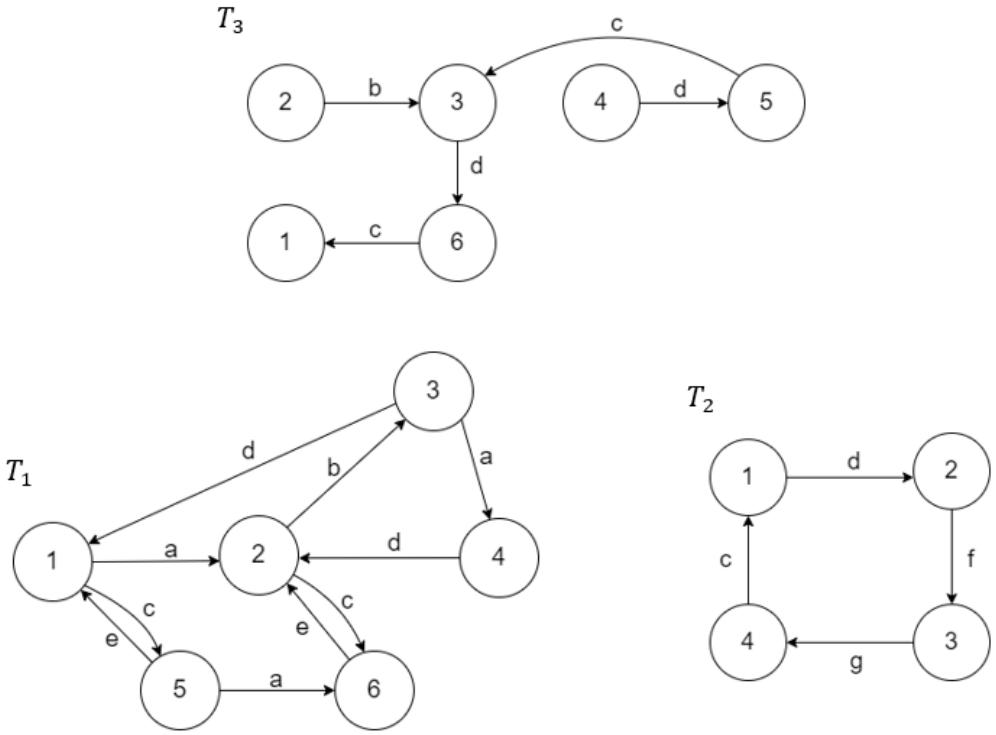

Figure 4.1: Composed attractor example.

4.2 Strong Composed Attractor

The notion of an invariant set is required to define the concept of strong state attraction as described in Section 2.8. Similarly, the notion of a composed invariant set is the basis for introducing the strong composed attractor in Definition 7.

Definition 7. *Let T_i, Σ and p_i, $i = 1, \ldots, l$ be given as in Definition 6. Let $C \subseteq W_1 \times \cdots \times W_l$ and $A \subseteq C$ be composed invariant sets. Then, A is denoted as a strong composed attractor for C in $T_1, \ldots, T_l$ if for all $c = (c_1, \ldots, c_l) \in C$*

1. *$\exists u \in \Sigma^\star$ and $a = (a_1, \ldots, a_l) \in A$ such that $\omega_i(c_i, p_i(u)) = a_i$ for all $i \in 1, \ldots, l$.*
2. *$\exists N$ such that $|u| < N$ for all u that fulfill (1).*

Definition 7 is a straightforward extension of Definition 2 to the case where the automaton under consideration is represented by the synchronous composition of multiple automata. Specifically, condition (1) in Definition 7 requires that, starting from a composed state $c = (c_1, \ldots, c_l)$ (each T_i starts from c_i), there must be a string u that moves each automaton T_i to its component a_i of a state in A. That is, all T_i, $i = 1, \ldots, l$ jointly move to the set A. In addition, (2) requires that any such string is not longer than a bound N. That is, starting from any state in C, the automata T_i, $i = 1, \ldots, l$ will jointly move to A with a bounded number of transitions.

We can consider again the example in Figure 4.1 with the same composed state set C as in Section 4.1. $A = \{\texttt{(1,1,1)}\}$ is a strong composed attractor for C in T_1, T_2 and T_3 because from any state in C, T_1, T_2 and T_3 will jointly reach to A in a bounded number of transitions. For example, from the composed state $(3, 1, 3)$, the string $s = \texttt{d f g c e}$ leads to the set A. In the scope of this thesis, the composed attractor verification was implemented as a function in the C++ software library libfaudes [41].

4.3 Supervisor Computation

According to Section 4.1 and 4.2, it is possible to verify the conditions of a composed invariant set and a strong composed attractor. Nevertheless, the missing important step is to synthesize supervisors such that these conditions can be applied in the scope of modular and abstraction-based supervisory control as described in Section 2.7. Specifically, the task to be accomplished can be described with the help of Figure 4.2.

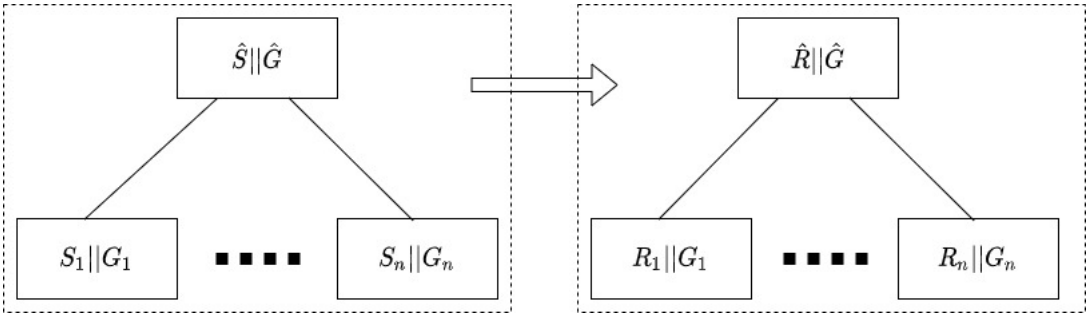

Figure 4.2: Switching from nonblocking control to state attraction.

Assume a nonblocking supervisory control loop has been designed for a DES accord-

ing to the procedure in Section 2.7. That is, there are n plant components $G_1, \ldots, G_n$ and their corresponding low-level supervisors $S_1, \ldots, S_n$. In addition, it is assumed that abstractions $\hat{G}_1, \ldots, \hat{G}_n$ of the low-level closed loops $S_1||G_1, \ldots, S_n||G_n$ are computed using natural projections $p_1, \ldots, p_n$ that fulfill the natural observer condition. The resulting abstracted plant is $\hat{G} = \hat{G}_1|| \cdots \hat{G}_n$ and $\hat{S}$ is a nonblocking supervisor for $\hat{G}$. That is, the overall nonblocking closed loop system is represented by $\hat{S}||S_1|| \cdots ||S_n||G_1|| \cdots ||G_n$.

If the DES evolves according to this closed loop, it is clear that the system will always be in a state that is described as a tuple $(x_1, \ldots, x_n)$, where $x_i \in X_i$ is a state of the automaton G_i for $i = 1, \ldots, n$. Now assume that it is desired to move the system to a pre-selected state $(a_1, \ldots, a_n)$ or a pre-specified state set $A \subseteq X_1 \times \cdots \times X_n$. This is for example necessary in the scope of reconfigurable manufacturing systems (RMS) or in the fault-tolerant control (FTC) of DES. Considering RMS, the system generally evolves according to some system configuration. If a change of the configuration is required, the operations of the current configuration need to be completed and the new configuration has to be initialized. This can be described by moving to a specific state or state set of the DES. Hence, the supervisory control of the DES has to switch to a different controller that realizes the described state attraction as illustrated in Figure 4.2. Considering FTC, it can be assumed that the DES is evolving according to its desired operation as long as there is no fault. If a fault occurs, the DES should either move to a system state, where the DES is safe or where it can continue its operation with reduced performance. Again, it is required to apply state attraction.

Accordingly, we next focus on the design of abstraction-based supervisors for state attraction for modular systems with multiple plant components $G_1, \ldots, G_n$. Hereby, we intend to address the following conditions:

1. We want to design low-level supervisors R_i for each plant component G_i, $i = 1, \ldots, n$ and a supervisor $\hat{R}$ for the abstracted plant $\hat{G}$,
2. We want to use the same state space as the supervisors $S_1, \ldots, S_n$ and $\hat{S}$ for the supervisor design for state attraction.

Condition 1. is beneficial in order to make use of the concept of abstraction-based

supervisor design to avoid the state space explosion problem. Condition 2. is introduced because of practical reasons. If this condition is fulfilled, it is possible to simple switch from the supervisors $S_1, \ldots, S_n, \hat{S}$ to the supervisors $R_1, \ldots, R_n, \hat{R}$. After this discussion, it is now possible to present the proposed design procedure:

1. Compute the supervisors $S_1, \ldots, S_n$ according to the desired system operation. It is assumed that the desired low-level system operation is represented by low-level specification automata $C_1, \ldots, C_n$.

2. Determine projections $p_1, \ldots, p_n$ for the system abstraction. Instead of computing natural observers as described in Section 2.6.3, we compute attraction-preserving observers as described in Section 3.2.

3. Compute a supervisor $\hat{S}$ for the abstracted plant $\hat{G} = \hat{G}_1 || \cdots \hat{G}_n$ according to the desired system operation. It is assumed that the desired system operation for the abstracted plant is represented by a specification automaton $\hat{C}$.

4. Compute a state-feedback supervisor for state attraction $\hat{R}$ for the abstracted plant $\hat{G}$ using a target set $\hat{A}$.

5. Use the supervisors for state attraction $\hat{R}_i = H_{S_i}$ for each $i = 1, \ldots, n$.

6. Obtain the overall closed loop under state attraction $\hat{R}\lceil(R_1 || \cdots || R_n)$.

The critical step in this procedure is step 2. Here, the natural projections $p_1, \ldots, p_n$ are attraction-preserving natural observers. That is, these projections are suitable for both the supervisor design for the normal system operation in step 1. to 3. and also for the state-feedback supervisor design for state attraction. As a result, the supervisors $S_1, \ldots, S_n$ can be directly used for the state attraction by computing $H_{S_1}, \ldots, H_{S_n}$ in step 5.

The operation $\lceil$ is defined in Section 3.3 for a single automaton G. In step 6., this automaton is given by the synchronous composition $H_{S_1} || \cdots || H_{S_n} = R_1 || \cdots || R_n$. Nevertheless, it is not desired to evaluate this synchronous composition to avoid the state space explosion. Accordingly, we next show how the operation $\lceil$ can be applied to each of the automata R_i, $i = 1, \ldots, n$, separately. To this end, we first recall that the abstracted automaton is computed as $\hat{G} = \hat{S}_1 || \cdots || \hat{S}_n$ with the abstractions of

the low-level supervisors $S_1, \ldots, S_n$. That is, each state $\hat{x} \in \hat{X}$ of $\hat{G}$ corresponds to an n-tuple of states from $\hat{S}_1, \ldots, \hat{S}_n$ and hence can be written as $\hat{x} = (\hat{q}_1, \ldots, \hat{q}_n)$. The same is true for each state $\hat{q} \in \hat{Q}$ of $\hat{S}$ since $\hat{S}$ is a state-feedback supervisor for $\hat{G}$. In addition, we know that $\hat{S}_1, \ldots, \hat{S}_n$ are computed using the natural projections $p_1, \ldots, p_n$ that all fulfill the attraction-preserving natural observer condition. That is, each p_i, $i = 1, \ldots n$ induces an equivalence relation μ_i and a corresponding canonical projection $cp_{\mu_i} : Q_i \to \hat{Q}_i$. This equivalence relation is the same for S_i and R_i by construction. Then, we define the supervisor action at each state $\hat{q} = (\hat{q}_1, \ldots, \hat{q}_n) \in \hat{Q}$ for the automaton R_i, $i = 1, \ldots, n$ as follows: Consider the states in the equivalence class $E_i \in Q_i/\mu_i$ such that $cp_{\mu_i}(E_i) = \hat{q}_i$. Then, all events in $(\Sigma_i \setminus \hat{\Sigma}_i) \cup (\hat{\Sigma}(\hat{q}) \cap \Sigma_i)$ are enabled at each state $q \in E_i$. The reason for this choice, is that, whenever $\hat{S}$ is in a state $\hat{q} = (\hat{q}_1, \ldots, \hat{q}_n)$, each automaton R_i must be in one of the states of the equivalence class E_i that belongs to the state $\hat{q}_I$. At each of these states, the control action of $\hat{S}$ that is relevant for R_i is applied, whereby all the low-level events are enabled.

We next show that state-attraction is achieved following the proposed procedure and applying the described supervisor implementation.

Theorem 5. *Consider the proposed procedure for abstraction-based state attraction for modular systems. Assume that $A \subseteq Q_1 \times \cdots \times Q_n$ is a composed invariant set such that $E_{1,\text{ex}} \times \cdots \times E_{n,\text{ex}} \subseteq A$ for some set of exit states $E_{i,\text{ex}}$ for each $i = 1, \ldots, n$. Let $\hat{A} = \{(cp_{\mu_1}(a_1), \ldots, cp_{\mu_n}(a_n)) | (a_1, \ldots, a_n) \in A\}$ and $\hat{R}$ be a state-feedback supervisor for $\hat{G}$ such that $\hat{A}$ is a strong attractor in $\hat{R}$. Then, A is a strong composed attractor in $\hat{R} \lceil \rceil (R_1 || \cdots || R_n)$.*

Theorem 5 starts with the assumption that it is desired to compute a state-feedback supervisor for state attraction with the target set A for a composed system with the plant components $G_1, \ldots, G_n$. Hereby, it is assumed that the canonical product of exit state sets from the different plant components is included in A. This assumption is motivated by the fact that it is always possible to reach some exit state when entering an equivalence class according to the construction of the automaton H_G. Then, the procedure is to compute a state-feedback supervisor $\hat{R}$ for the abstracted plant $\hat{G}$ and apply this supervisor to the synchronous composition of the automata $R_i = H_{S_i}$ of

the different plant components $i = 1, \ldots, n$. The main result of the theorem is that the target set A is a strong attractor for the overall closed-loop system $\hat{R}\lceil\rceil(R_1||\cdots||R_n)$. We next prove the theorem. Illustrative examples are provided in the next section.

Proof. In order to prove the theorem, we assume that the closed-loop system $\hat{R}\lceil\rceil(R_1||\cdots||R_n)$ is in an arbitrary state given by the tuple $(q_1, \ldots, q_n, \hat{q})$. We further write $R = (Q, \Sigma, \nu, q_0, Q_\mathrm{m}) = R_1||\cdots||R_n$. For convenience, we also introduce the automaton $R = (Q, \sigma, \nu, q_0, Q_\mathrm{m}) = R_1||\cdots||R_n$.

Since $\hat{A}$ is a strong attractor for $\hat{Q}$ in $\hat{S}$, there is a string $v \in \hat{\Sigma}^\star$ with $|v| \leq \hat{N}$ for some $\hat{N} \in \mathbb{N}$ and such that $\hat{\nu}(\hat{q}, v) = \hat{q}' \in \hat{A}$. Since $p_1, \ldots, p_n$ are natural observers, it directly follows that there must be strings $u_i \in \Sigma_i^\star$ for $i = 1, \ldots, n$ such that $v \in ||_{i=1}^n u_i$. Writing $\hat{q}' = (\hat{q}'_1, \ldots, \hat{q}'_n)$, it must further be the case for each $i = 1, \ldots, n$ that $\nu_i(q_i, u_i) = q'_i \in E_i$ for some equivalence class E_i with $cp_{\mu_i}(E_i) = \hat{q}'_i$. That is, since $E_{i,\mathrm{ex}}$ is a strong attractor for E_i in G_{E_i} by construction, there exists a string $u'_i \in (\Sigma_i \setminus \hat{\Sigma}_i)^\star$ such that $\nu_i(q'_i, u'_i) \in E_{i,\mathrm{ex}}$ and $|u'_i| \leq N_i$ for some $N_i \in \mathbb{N}$. Furthermore, it holds that we can write u_i as $u_i = u_i^1\sigma_1 u_i^2\sigma_2 \cdots u_i^m\sigma_m$, where $\sigma_1\sigma_2 \ldots \sigma_m$ is the sequence of high-level events in v that belong to Σ_i. Since $|v| \leq \hat{N}$, also $|\sigma_1\sigma_2 \cdots \sigma_m| \leq \hat{N}$. Moreover, it holds for each u_i^k, $k = 1, \ldots, m$ that $|u_i^k| \leq N_k$ for some $N_k \in \mathbb{N}$ since each u_i^k only passes equivalence classes that do not have any cycles by construction. Together, we constructed strings $u_i u'_i \in \Sigma_i^\star$ for each $i = 1, \ldots, n$ such that $\nu_i(q_i, u_i u'_i) \in E_{i,\mathrm{ex}}$ and hence it holds for each string $u \in ||_{i=1}^n$ that $\nu(q, u) \in E_{1,\mathrm{ex}} \times \cdots \times E_{n,\mathrm{ex}} \subseteq A$. Since $u_i u'_i$ is bounded for each $i = 1, \ldots, n$, also there must be a bound $N \in \mathbb{N}$ such that $|u| \leq N$. This implies that A is a string composed attractor in $\hat{R}\lceil\rceil(R_1||\cdots||R_n)$. □

4.4 Illustrative Example

This section provides an illustrative example to explain the supervisor computation in the previous section. To this end, we first point out one more issue and its solution in Section 4.4.1. After that, we demonstrate the supervisor computation for a system with two components.

4.4.1 Low-Level Supervisor Computation

The automaton H_G in Section 3.1 is computed to achieve state attraction in the low level. Hereby, the conditions on H_G require disabling of low-level cycles with low-level events since they cannot be disabled by the computed abstracted supervisor. The abstracted supervisor can only take care of cycles with high-level events. However, not every supervisor is suitable for state attraction. A supervisor might not be suitable for state attraction if not all its states are included in the optimal set for state attraction. Figure 4.3 shows an low-level example supervisor $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$ which is not suitable for state attraction. Specifically, there is a cycle with an uncontrollable event `u1` between the states 7 and 9. This implies that whenever state 7 or 9 are reached, the automaton might cycle between these states forever. Accordingly, we get the optimal set for state attraction $\Omega_G(A) = \{1,2,3,5,6,8,10,11,12,13,14,15,16,17,18\}$ for the set $A = \{1\}$, which does not include the states 7 and 9. That is, in this case, it is also clear that H_G cannot be computed as desired since $\Omega_G(A) \neq X$.

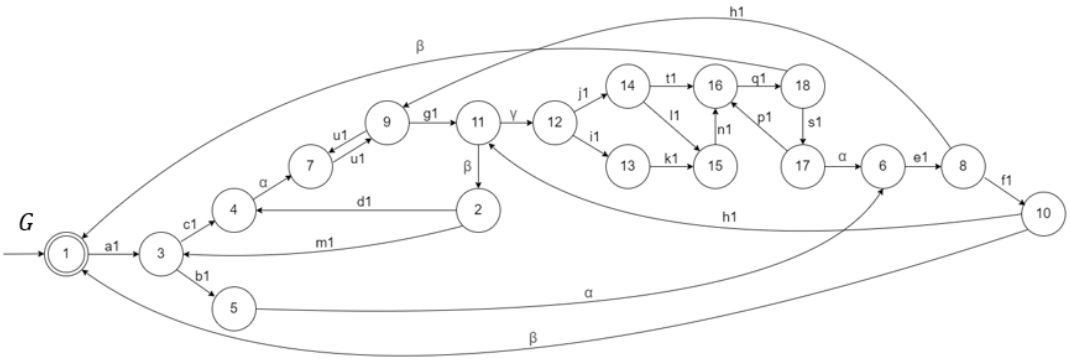

Figure 4.3: Example low-level supervisor G not suitable for state attraction.

In this case, our aim is to find a modified low-level supervisor which is suitable for state attraction. The basic algorithm for finding such modified low-level supervisor is given as follows, assuming that an automaton G and attractive set A are given:

1. Compute $\Omega_G(A)$
2. Compute subautomaton $G' \sqsubset G$ by removing states that do not belong to $\Omega_G(A)$ from G

3. Compute modified low-level supervisor G_{m} which is suitable for state attraction using G' as specification for the $SupC$ algorithm.

The subautomaton G' that fulfills state attraction and the corresponding modified low-level supervisor G_{m} are given in Figure 4.4 and Figure 4.5, respectively. G_{m} is suitable for state attraction but it is not an attractive supervisor yet. It will be used in the normal operation of the system, whereby $H_{G_{\mathrm{m}}}$ will be used for state attraction.

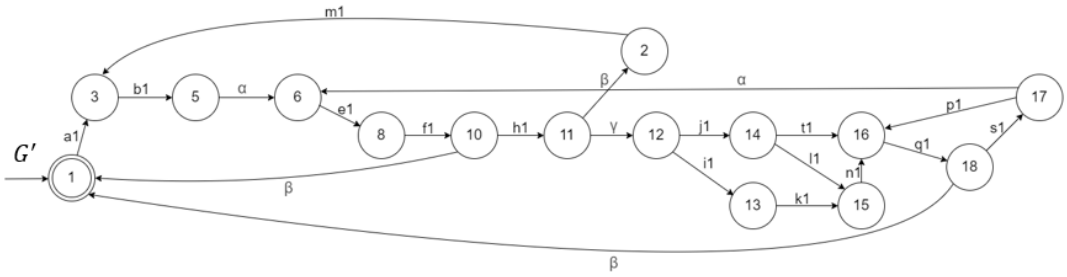

Figure 4.4: Subautomaton G' that fulfills weak attractor.

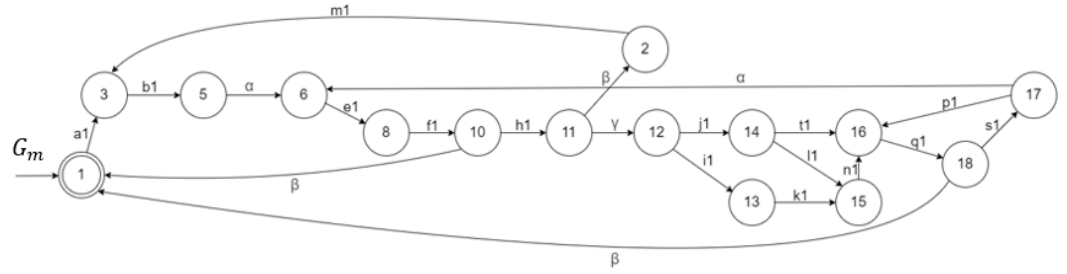

Figure 4.5: Modified low-level supervisor G_m.

Moreover, if an abstraction is not an attraction-preserving natural observer, it is not suitable for abstraction-based state attraction. If we consider the modified low-level supervisor G_{m} in Figure 4.5 with abstraction alphabet $\hat{\Sigma} = \{\alpha, \beta, \gamma, \mathtt{s1}, \mathtt{p1}\}$, the states 12, 13, 14, 15, 16, 17 and 18 are in the same equivalence class and the states 17 and 18 are exit states. Their outgoing abstraction events are $\hat{\Sigma}_{17} = \{\alpha\}$ and $\hat{\Sigma}_{18} = \{\beta\}$. In this case, $(17, 18)$ is a bad pair since different abstractions events are possible from these exit states and 17 and 18 should not be in the same equivalence class. Therefore, the projection with $\hat{\Sigma}$ is not an attraction-preserving natural observer and it is not suitable for state attraction. Another bad exit state pair is $(10, 11)$ because the states 6, 8, 10 and 11 are in the same equivalence class and the states 10 and

11 are exit states. Their outgoing abstraction events are $\hat{\Sigma}_{10} = \{\beta\}$ and $\hat{\Sigma}_{11} = \{\beta, \gamma\}$. In this case, $\hat{\Sigma}_{10} \subset \hat{\Sigma}_{11}$ and since $10 \notin \Omega_{G_m}(\{11\})$, $(10, 11)$ is a bad pair. $(17, 18)$ can be split by adding event $a1$; $(10, 11)$ can be split by adding event $h1$ to the abstraction alphabet $\hat{\Sigma}$. That is extending the abstraction alphabet to $\hat{\Sigma} = \{\alpha, \beta, \gamma, \texttt{a1}, \texttt{h1}\}$ will split these bad pairs. The resulting abstraction of G_{m} with $\hat{\Sigma} = \{\alpha, \beta, \gamma, \texttt{s1}, \texttt{p1}, \texttt{h1}, \texttt{a1}\}$ is given in Figure 4.6.

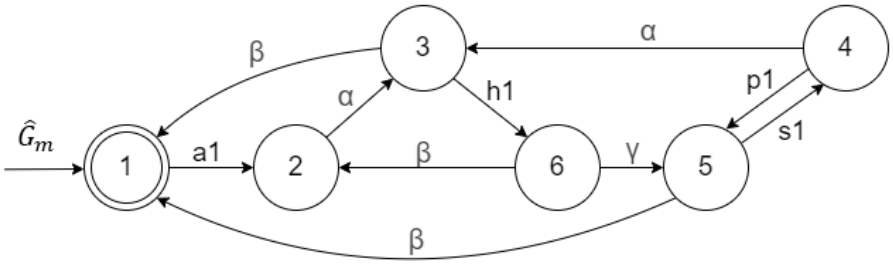

Figure 4.6: Abstraction of modified low-level supervisor G_m.

After computing the modified low-level supervisor G_{m} and finding a suitable abstraction alphabet $\hat{\Sigma}$, we can compute the automaton $H_{G_{\mathrm{m}}}$ by disabling low-level cycles in G_{m}, We recall that it is not required to remove cycles with abstraction events since the abstracted supervisor for state-attraction will achieve this. Since G_{m} in Figure 4.5 does not have any low-level cycles, it holds that $H_{G_{\mathrm{m}}} = G_{\mathrm{m}}$ for this example.

4.4.2 Supervisor Computation for a Composed System

After the preparation in the previous section, we now apply the full abstraction-based supervisor computation for state attraction to an example system with the two low-level supervisors S_1 and S_2 in Figure 4.7. The desired abstraction alphabet is given as $\hat{\Sigma} = \{\alpha, \beta, \gamma, \delta, \phi\}$. We note that these supervisors can be considered as low-level supervisors for the normal operation of an RMS.

Our aim is to first compute the supervisors for the normal operation of the DES by computing an abstraction-based supervisor. Then, we want to compute low-level and abstraction-based supervisors for state-attraction. First, we need to compute the modified low-level supervisors which are suitable for state attraction. Figure 4.8 shows the modified low-level supervisors S_{m1} and S_{m2} that will be used for low-level attrac-

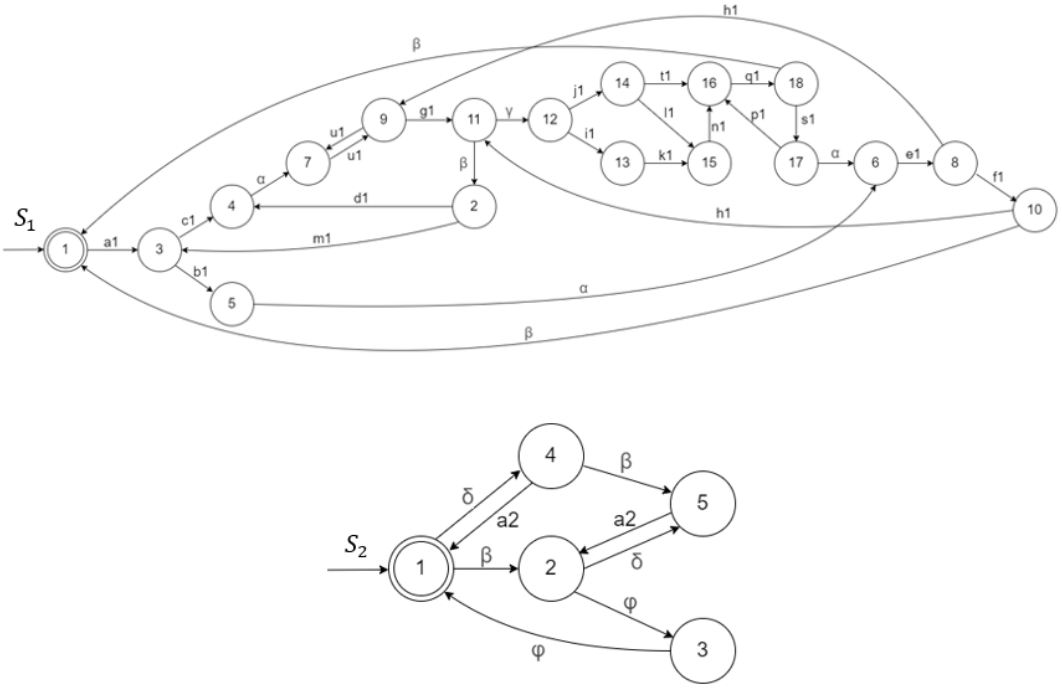

Figure 4.7: Example system with two low-level supervisors S_1 and S_2.

tor calculation. Hereby, it can be noted that S_{m1} is the supervisor considered in the previous section, whereas no modification is required for S_2 such that $S_{\mathrm{m2}} = S_2$.

Next, using the modified low-level supervisors, an attraction-preserving natural observer for each supervisor is computed as in described in Section 3.2. Figure 4.9 shows the abstraction of S_{m1} and S_{m2} as $\hat{S}_1$ and $\hat{S}_2$.

Then, using the synchronous composition $\hat{G} = \hat{S}_1||\hat{S}_2$ as the abstracted plant and the specification C, which is given in Figure 4.10 compute an abstracted supervisor $\hat{S}$ such that $L_{\mathrm{m}}(\hat{S}) = SupC(\hat{G}, L_{\mathrm{m}}(C), \hat{\Sigma}_{\mathrm{u}})$. The computed supervisor $\hat{S}$ is shown in Figure 4.11. That is, the normal operation of the DES (which can be considered as an RMS) is given by applying the supervisors $\hat{S}||S_1||S_2$.

Up to this point, the supervisors for the normal operation of the DES were computed. Next, we want to compute the supervisors for state attraction. Assume, we want to move to the target set $A = \{(1, 1)\}$, which could be considered as the start state of a new configuration of an RMS. That is, we want to move to the initial state of each of the low-level supervisors S_1 and S_2. Applying the canonical projection, this

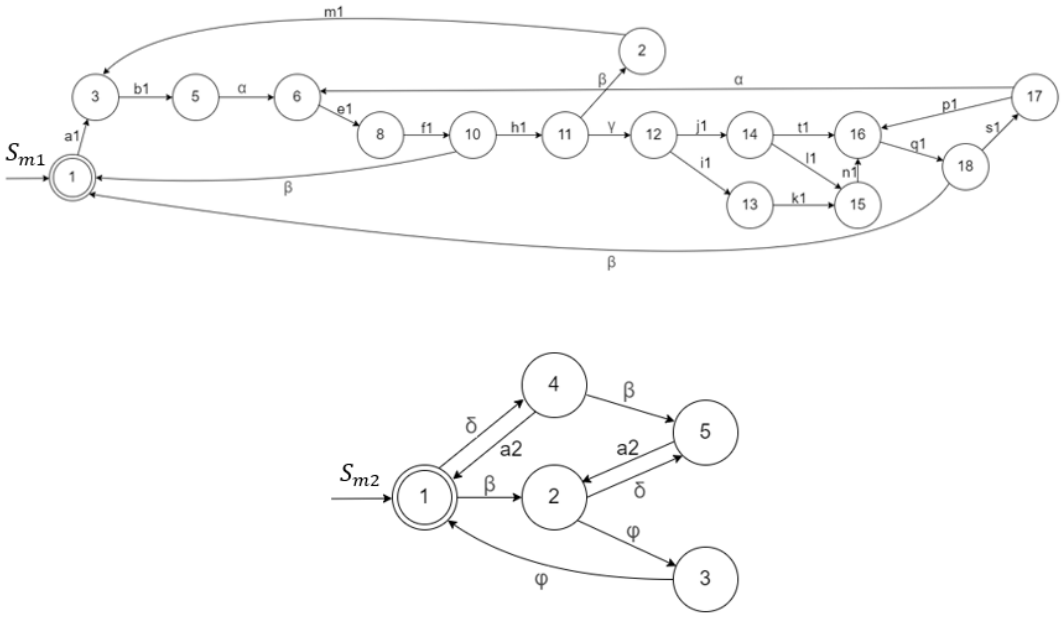

Figure 4.8: Modified low-level supervisors S_{m1} and S_{m2}.

corresponds to the target set $\hat{A} = \{1\}$ in $\hat{S}$. That is, we now compute a minimally restrictive optimal supervisor for state attraction as in [39] for $\hat{S}$. The resulting supervisor for state attraction $\hat{R}$ is shown in Figure 4.12.

Finally, the low-level supervisors R_1 and R_2 are computed as shown in Figure 4.13.

In addition, for clarity, all the supervisors $\hat{R}$, R_1 and R_2 that are active during state attraction are depicted in Figure 4.14. It can be seen that $\hat{R}$ is going to move the RMS from each state to the attractive set A when a reconfiguration switch is requested.

We finally describe a detailed example scenario with our example system. We recall that Figure 4.8 shows the low-level supervisors and Figure 4.11 shows the abstraction-based supervisor for the normal operation of the example system. Then, we consider the case where the RMS is at the states $(9, 1)$ of S_{m1} and S_{m2} during the normal system operation and the abstraction-based supervisor $\hat{S}$ is at the corresponding state 9. When performing a configuration change, we want to move the RMS to the initial state, which is described by the state triple $(1, 1, 1)$. Hence, we now directly switch to the supervisors for state attraction in Figure 4.14 starting from the current state triple $(9, 1, 9)$. That is, R_1 is at state 9, R_2 is at state 1 and $\hat{R}$ is at state 9. Then, the abstraction-based supervisor $\hat{R}$ can move to the initial state with the string $\hat{s} = \beta\phi\phi$.

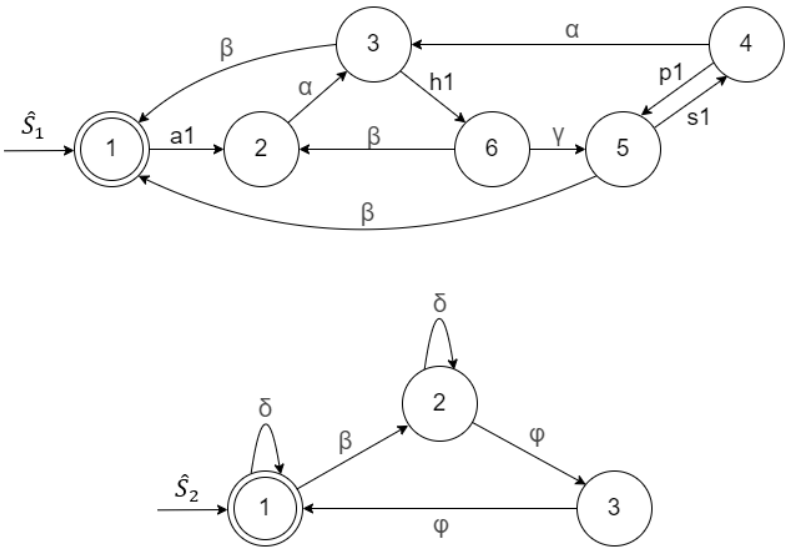

Figure 4.9: Abstraction of S_{m1} and S_{m2}.

The corresponding strings in the low level are $s_1 = j1t1q1\beta$ in R_1 and $s_2 = \phi\phi$ in R_2. That is, jointly following these strings, the RMS also moves to initial state in the low-levels. In detail, when starting at $(9, 1)$, supervisor S_1 can first evolve independently from state 9 to state 14 with the string of low-level events `j1 t1 q1`. All these events are enabled by the definition of $\hat{R}\lceil\rceil(R_1||R_2)$ and $\hat{R}$ and R_2 do not change state. That is, the corresponding state triples are $(9, 1, 9)$, $(11, 1, 9)$, $(13, 1, 9)$, $(14, 1, 9)$. In the state triple $(14, 1, 9)$, the shared event β is possible in all supervisors. Hence, the RMS moves to the triple $(1, 2, 11)$ with β. Then, R_1 is already in the desired state, whereas R_2 and $\hat{R}$ jointly move from $(1, 2, 11)$ to $(1, 3, 12)$ with φ and to $(1, 1, 1)$ with φ, which is the target state for state attraction.

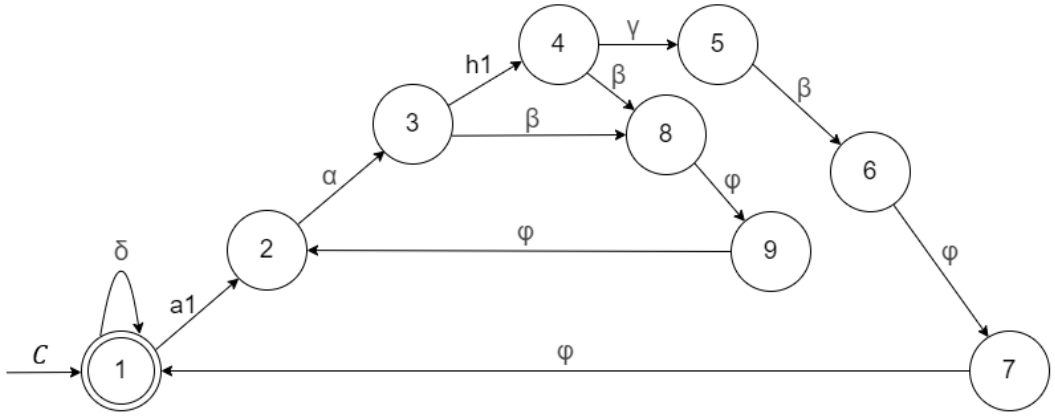

Figure 4.10: Specification C.

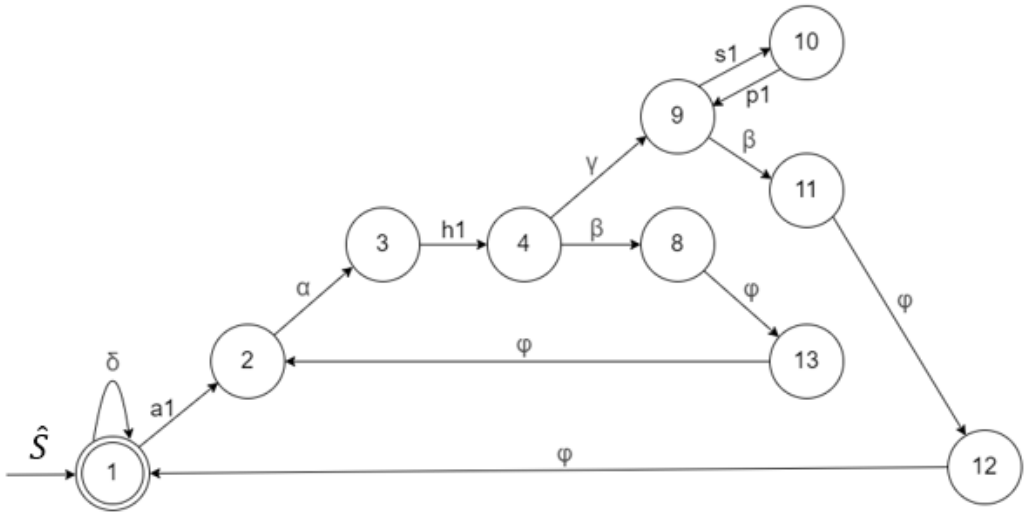

Figure 4.11: High-level supervisor $\hat{S}$.

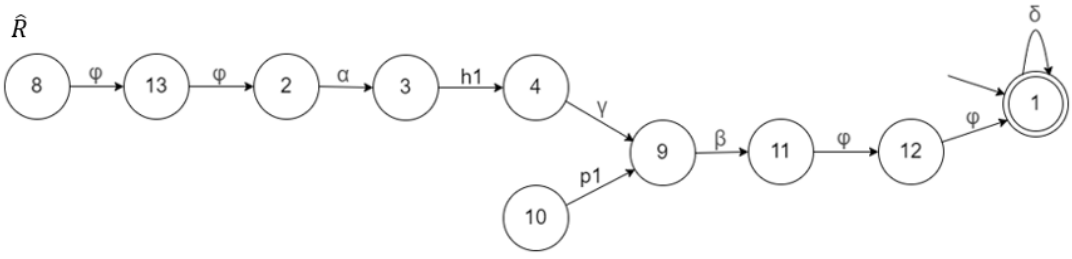

Figure 4.12: High-level attractor $\hat{R}$.

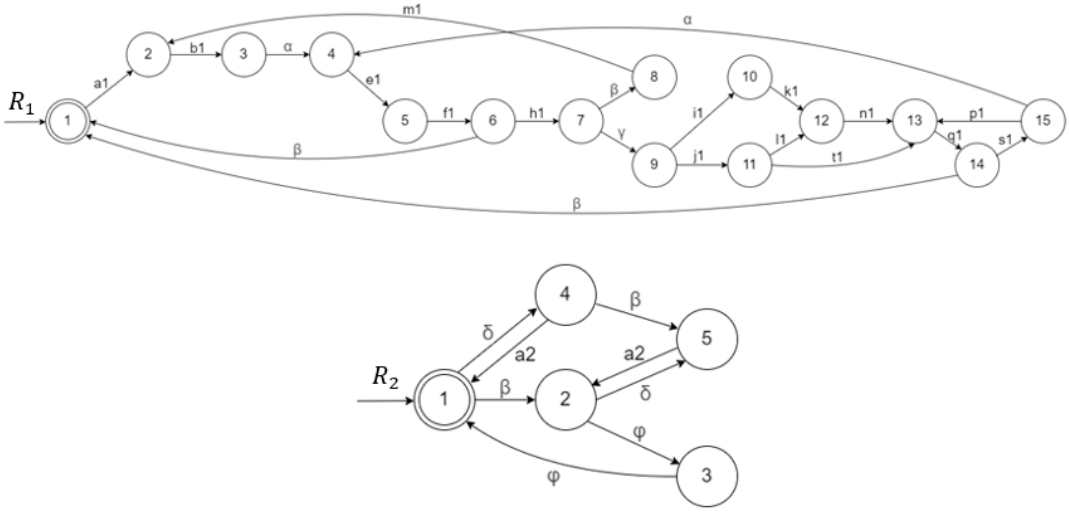

Figure 4.13: Low-level attractors R_1 and R_2.

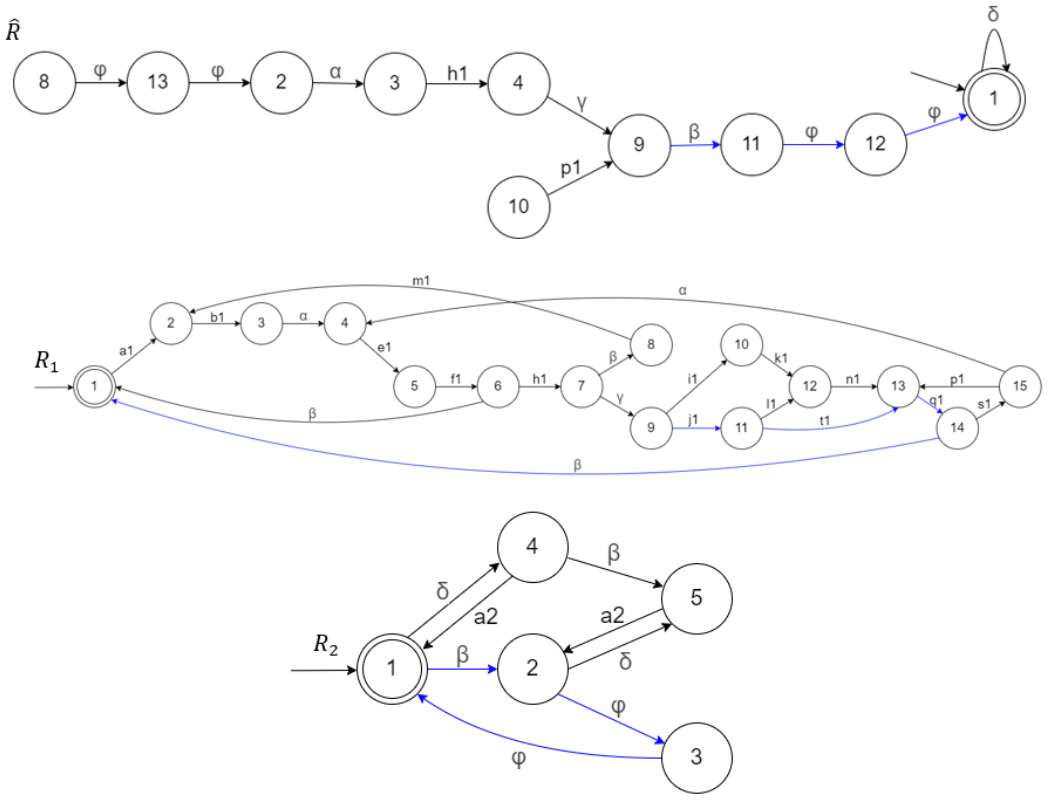

Figure 4.14: Supervisors for the example system.

4.5 Limitations: Maximal Permissiveness

The previous sections develop the supervisor computation method and includes an illustrative example. However, it has to be mentioned that is not always possible to compute such supervisors. Maximal permissiveness can be the limitation of the supervisor computation method because if the abstraction-based supervisor architecture is not maximally permissive, the closed loop with the abstraction-based supervisor will not be the same as the closed loop with a monolithic supervisor.

In order to illustrate this limitation, we can think of the example with the low-level plant G, which is given in Figure 4.15 with controllable events $\Sigma_c = \{\texttt{d}, \texttt{f}, \texttt{h}, \texttt{i}, \texttt{j}, \beta\}$ and abstraction alphabet $\hat{\Sigma} = \{\alpha, \beta, \gamma, \delta\}$.

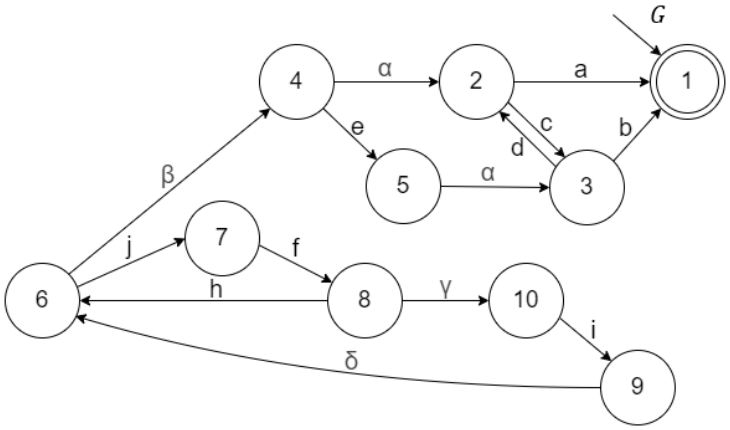

Figure 4.15: Example plant G.

Figure 4.16 shows the abstraction $\hat{G}$ and the state-feedback supervisor $\hat{S}$ for the high-level and for $\hat{A} = \{1\}$ is shown in Figure 4.17. It can be seen that transition β has to be disabled since there is a loop with γ and δ, which are both uncontrollable events and they cannot be disabled.

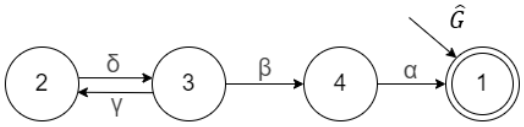

Figure 4.16: Abstraction $\hat{G}$.

Figure 4.18 shows computed state-feedback supervisor S for the low-level and for

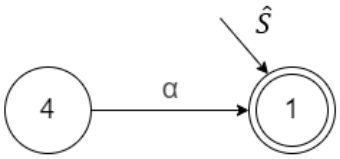

Figure 4.17: State-feedback supervisor $\hat{S}$ for the high-level and for $\hat{A} = \{1\}$.

$A = \{1\}$.

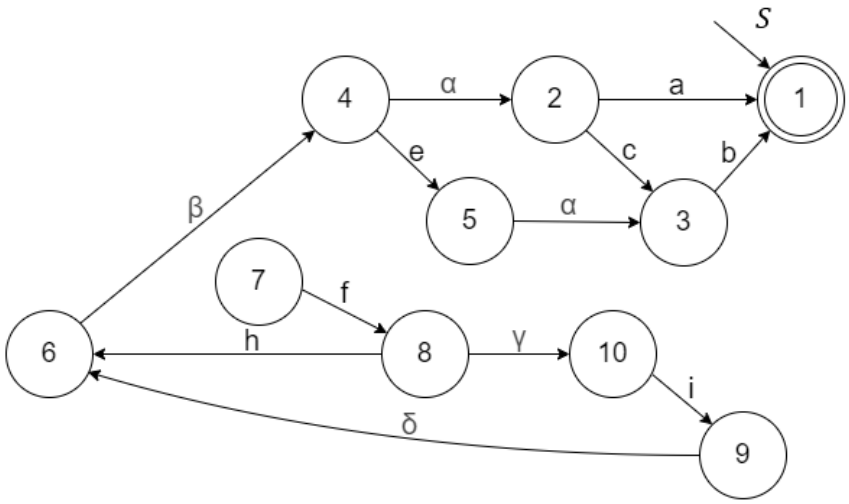

Figure 4.18: State-feedback supervisor S for the low-level and for $A = \{1\}$.

However, if we compute abstraction-based state-feedback supervisor according to Definition 5, this leads to a smaller attractor in the low-level for our example. The overall closed loop $T = \hat{S}||S||G$ can be seen in Figure 4.19.

In summary, the abstraction-based supervisor architecture is not maximally permissive. That is, it is possible that a larger set of weak attraction can be found in the monolithic case compared to the abstraction-based case. This is not necessarily a problem if a non-empty supervisor with a sufficient number of states can be computed. Nevertheless, it is undesired if the weak set of state attraction does not contain all the desired states. In order to overcome this problem, in the future work, local control consistency should be checked [38]. It is known from the nonblocking hierarchical and modular supervisory control that local control consistency is sufficient for maximally permissive control. That is, it would be interesting to investigate if

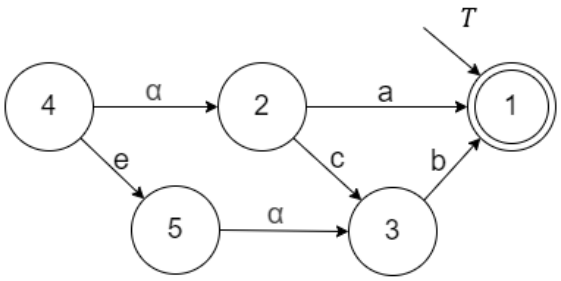

Figure 4.19: The overall closed loop T.

the same condition is also suitable for maximally permissive abstraction-based state attraction.

CHAPTER 5

CONCLUSION

Reconfigurable manufacturing systems (RMS) represent a new manufacturing idea that allows manufacturing systems to quickly modify their production capacity and functionality. In principle, the RMS controller design should allow users to switch between more than one system configuration at any moment. Whenever a switch is requested by the user, it is expected that the overall system should first finish the active configuration and then start the requested configuration.

In this thesis, a new controller design methodology that supports modular design and is scalable to large-scale RMS is developed. Specifically, modular and abstraction-based control approaches for state attraction are devised and then applied to an example for illustration. The thesis first summarizes the required basic notations about discrete event systems, supervisory control theory, abstraction-based control using natural observers and state attraction. When a reconfiguration is requested, an RMS should finish the current configuration before starting the newly requested configuration. A supervisor for state attraction can be used to accomplish this task. However, supervisors for state attraction cannot be used for large-scale RMS due to the state space explosion problem. As a first step to address this problem, the thesis develops a new method for abstraction-based state attraction. This method is based on the notion of an attraction-preserving natural observer, which is also introduced in the thesis. Furthermore, the thesis provides an algorithm for computing attraction-preserving natural observers and applies it to different examples.

It has to be noted that this method is only suitable for monolithic systems and hence still not fully applicable to large-scale RMS. In order to accomplish state attraction for large-scale systems, the proposed method is extended to composed state attrac-

tion. That is, we compute low-level supervisors for modular system components that are then abstracted using attraction-preserving natural observers. Finally, a supervisor for state attraction is computed for the abstracted system and then applied to all the modular components. The suitability of this method is demonstrated by various examples. All presented algorithms are implemented in the open-source C++ library for DES libfaudes.

In future work, it is intended to study the suitability of the developed method in multi-level hierarchies and to extend the method to maximally permissive abstraction-based state attraction.

REFERENCES

[1] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems, Second edition*. Springer, 2008.

[2] W. M. Wonham, "Supervisory control of discrete-event systems," *Department of Electrical and Computer Engineering, University of Toronto*, 2021.

[3] K. Cai and W. Wonham, "Supervisory control of discrete-event systems," in *Encyclopedia of Systems and Control*, Springer London, UK, 2020.

[4] R. Wisniewski, M. Zhou, L. Gomes, M. P. Fanti, and R. Kumar, "Special issue on recent advances in petri nets, automata, and discrete-event hybrid systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 10, pp. 3484–3487, 2020.

[5] B. Caillaud, P. Darondeau, L. Lavagno, and X. Xie, *Synthesis and control of discrete event systems*. Springer Science & Business Media, 2002.

[6] C. Seatzu, M. Silva, and J. H. Van Schuppen, *Control of discrete-event systems*, vol. 433. Springer, 2013.

[7] M. Iordache and P. J. Antsaklis, *Supervisory control of concurrent systems: a Petri net structural approach*. Springer Science & Business Media, 2007.

[8] 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.

[9] 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.

[10] K. Schmidt, T. Moor, and S. Perk, "Nonblocking hierarchical control of decentralized discrete event systems," *Automatic Control, IEEE Transactions on*, vol. 53, no. 10, pp. 2252–2265, 2008.

[11] K. Schmidt and C. Breindl, "Maximally permissive hierarchical control of decentralized discrete event systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 4, pp. 723–737, 2011.

[12] J. Campos, C. Seatzu, and X. Xie, *Formal methods in manufacturing*. CRC press, 2018.

[13] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. V. Brussel, "Reconfigurable manufacturing systems," *CIRP Annals – Manufacturing Technology*, vol. 48, pp. 527–540, 1999.

[14] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *Journal of Intelligent Manufacturing*, vol. 11, pp. 403–419, 2000.

[15] H. A. ElMaraghy, *Changeable and Reconfigurable Manufacturing Systems*. Springer Series in Advanced Manufacturing, 2009.

[16] Y. Koren, *The global manufacturing revolution*. Wiley, 2010.

[17] R. Sampath, H. Darabi, U. Buy, and L. Jing, "Control reconfiguration of discrete event systems with dynamic control specifications," *Automation Science and Engineering, IEEE Transactions on*, vol. 5, no. 1, pp. 84 –100, 2008.

[18] J. Li, X. Dai, and Z. Meng, "Automatic reconfiguration of petri net controllers for reconfigurable manufacturing systems with an improved net rewriting system-based approach," *Automation Science and Engineering, IEEE Transactions on*, vol. 6, no. 1, pp. 156 –167, 2009.

[19] G. Faraut, L. Piétrac, and E. Niel, "Formal approach to multimodal control design: Application to mode switching," *Industrial Informatics, IEEE Transactions on*, vol. 5, no. 4, pp. 443–453, 2009.

[20] F.-S. Hsieh, "Design of scalable agent-based reconfigurable manufacturing systems with petri nets," *International Journal of Computer Integrated Manufacturing*, vol. 31, no. 8, pp. 748–759, 2018.

[21] H. Kaid, A. Al-Ahmari, Z. Li, and R. Davidrajuh, "Automatic supervisory controller for deadlock control in reconfigurable manufacturing systems with dynamic changes," *Applied Sciences*, vol. 10, no. 15, p. 5270, 2020.

[22] H. Kaid, A. Al-Ahmari, and Z. Li, "Colored resource-oriented petri net based ladder diagrams for plc implementation in reconfigurable manufacturing systems," *IEEE Access*, vol. 8, pp. 217573–217591, 2020.

[23] H. E. Garcia and A. Ray, "State-space supervisory control of reconfigurable discrete event systems," *International Journal of Control*, vol. 63, no. 4, pp. 767–797, 1996.

[24] E. W. Endsley, E. E. Almeida, and D. M. Tilbury, "Modular finite state machines: Development and application to reconfigurable manufacturing cell controller generation," *Control Engineering Practice*, vol. 14, no. 10, pp. 1127 – 1142, 2006.

[25] K. W. Schmidt, "Optimal configuration changes for reconfigurable manufacturing systems," in *Decision and Control, IEEE Conference on*, pp. 7621–7626, 2013.

[26] K. Schmidt, "Computation of supervisors for reconfigurable machine tools," *Discrete Event Dynamic Systems*, vol. 25, no. 1-2, pp. 125–158, 2015.

[27] A. Nooruldeen and K. Schmidt, "State attraction under language specification for the reconfiguration of discrete event systems," *Automatic Control, IEEE Transactions on*, vol. 60, pp. 1630–1634, June 2015.

[28] H. M. Khalid, M. S. Kırık, and K. W. Schmidt, "Abstraction-based supervisory control for reconfigurable manufacturing systems," *IFAC Proceedings Volumes*, vol. 46, no. 22, pp. 157–162, 2013.

[29] R. Sengupta and S. Lafortune, "An optimal control theory for discrete event systems," *SIAM J. Control Optim.*, vol. 36, no. 2, pp. 488–541, 1998.

[30] Y. Brave and M. Heymann, "Stabilization of discrete-event processes," *Int. J. Control*, vol. 51, pp. 1101–1117, 1990.

[31] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Introduction to automata theory, languages, and computation," *Acm Sigact News*, vol. 32, no. 1, pp. 60–65, 2001.

[32] K. C. Wong and W. M. Wonham, "Hierarchical control of discrete-event systems," *Discrete Event Dynamic Systems*, vol. 6, pp. 241–273, 1996.

[33] R. J. Leduc, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control-part i: Serial case," *IEEE Transactions on Automatic Control*, vol. 50, no. 9, pp. 1322–1335, 2005.

[34] K. C. Wong and W. M. Wonham, "On the computation of observers in discrete-event systems," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 14, no. 1, pp. 55–107, 2004.

[35] K. Schmidt and T. Moor, "Marked-string accepting observers for the hierarchical and decentralized control of discrete event systems," in *Discrete Event Systems, Workshop on*, pp. 413–418, 2006.

[36] L. Feng and W. M. Wonham, "On the computation of natural observers in discrete-event systems," *Discrete Event Dynamic Systems*, vol. 20, no. 1, pp. 63–102, 2010.

[37] J.-C. Fernandez, "An implementation of an efficient algorithm for bisimulation equivalence," *Science of Computer Programming*, vol. 13, pp. 219–236, 1990.

[38] K. Schmidt and C. Breindl, "Maximally permissive hierarchical control of decentralized discrete event systems," *Automatic Control, IEEE Transactions on*, vol. 56, no. 4, pp. 723 –737, 2011.

[39] Y. Brave and M. Heymann, "On optimal attraction of discrete-event processes," *Information Sciences*, vol. 67, pp. 245–276, 1993.

[40] R. Kumar, V. Garg, and S. I. Marcus, "Language stability and stabilizability of discrete event dynamical systems," *SIAM J. Control Optim.*, vol. 31, pp. 132–154, 1993.

[41] T. Moor, K. Schmidt, and S. Perk, "libfaudes—an open source c++ library for discrete event systems," in *2008 9th International Workshop on Discrete Event Systems*, pp. 125–130, IEEE, 2008.