FAULT-TOLERANT SUPERVISORY CONTROL OF DISCRETE EVENT SYSTEMS: METHODS AND EXAMPLES

AYŞE NUR ACAR

AUGUST 2015

FAULT-TOLERANT SUPERVISORY CONTROL OF DISCRETE EVENT SYSTEMS: METHODS AND EXAMPLES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF
ÇANKAYA UNIVERSITY

BY

AYŞE NUR ACAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF
ELECTRONIC AND COMMUNICATION ENGINEERING

AUGUST 2015

Title of the Thesis: **Fault-Tolerant Supervisory Control of Discrete Event Systems: Methods and Examples**

Submitted by **AYŞE NUR ACAR**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.

Prof. Dr. Halil Tanyer EYYUBOĞLU
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Halil Tanyer EYYUBOĞLU
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Klaus Werner SCHMIDT
Supervisor

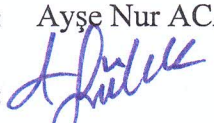
Examination Date: 12.08.2015

Examining Committee Members

Assoc. Prof. Dr. Orhan GAZİ	(Çankaya Univ.)	
Assoc. Prof. Dr. Klaus Werner SCHMIDT	(Çankaya Univ.)	
Assoc. Prof. Dr. Umut ORGUNER	(ODTÜ)	

STATEMENT OF NON-PLAGIARISM PAGE

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, Last Name	:	Ayşe Nur ACAR
Signature	:	
Date	:	12.08.2015

ABSTRACT

FAULT-TOLERANT SUPERVISORY CONTROL OF DISCRETE EVENT SYSTEMS: METHODS AND EXAMPLES

ACAR, Ayşe Nur

M.Sc., Department of Electronic and Communication Engineering

Supervisor: Assoc. Prof. Dr. Klaus Werner SCHMIDT

August 2015, 50 pages

Faults can be considered as uncontrollable events that suddenly happen in a system and change the behaviour of the system in a negative way. In particular, in case a fault happens in a discrete event system (DES), certain actions or operations might no longer be possible. This thesis studies the supervisory control of DES that are subject to faults. Hereby, two concepts are employed: Fault-tolerant and fault-recovery control.

Regarding fault-tolerant control, it is desired to use a controller that works both for the system with and without a fault. Hence, we first identify necessary and sufficient conditions for the existence of a supervisor that realizes a given behavioral specification both in the non-faulty and in the faulty case. We further show that it is possible to determine a supremal fault-tolerant sublanguage in case the existence condition is violated. Finally, we propose an algorithm for the computation of this sublanguage and prove

its correctness. Different from existing work, our fault-tolerant supervisor allows fault occurrences and system repairs at any time.

Regarding fault-recovery control, we study both the case of operating a DES after a fault and after repair. We first develop a new method for the fault-recovery of DES. In particular, we compute a fault-recovery supervisor that follows the specified nominal system behavior until a fault occurrence, that continues its operation according to a degraded specification after a fault and that finally converges to a desired behavior after fault. We next show that our method is also applicable to system repair and we propose an iterative procedure that determines a supervisor for an arbitrary number of fault occurrences and system repairs.

Finally, we extend our fault-recovery and repair method with multiple and different faults and repairs. As a result, we obtain a supervisor that follows the specified nominal system behavior in the fault-free case, converges to a desired degraded behavior for each fault type and recovers the nominal behavior after corresponding repair. All developed methods are demonstrated with a small manufacturing system.

Keywords: Discrete Event Systems, Supervisory Control, Fault, Fault-Tolerance, Fault-Recovery, Repair.

ÖZ

AYRIK OLAYLI SİSTEMLER İÇİN HATAYA DAYANIKLI DENETLEYİCİ KONTOL: METODLAR VE ÖRNEKLER

ACAR, Ayşe Nur
Yüksek Lisans, Elektronik ve Haberleşme Mühendisliği Anabilim Dalı
Tez Yöneticisi: Doç Dr. Klaus Werner SCHMIDT
Ağustos 2015, 50 sayfa

Hatalar kontrol edilemeyen olaylardır. Sistemde bir anda meydana gelirler ve sistemin çalışma düzenin negatif yönde etkilerler. Genellikle, ayrık olaylı sistemlerde (DES) hata olması durumunda, ilgili olayların veya operasyonların bir daha oluşmama ihtimali vardır. Bu tezde hata içeren ayrık olaylı sistemlerin denetleyici kontrolü çalışılmıştır. Bu vesile ile iki ana konsept incelenmiştir: Hataya dayanıklılık ve hata kurtarıcı kontrol.

Hataya dayanıklı kontrole ilişkin, sistemin hem normal durumunda hem de hatalı durumda tek bir kontrolör kullanılması amaçlanmıştır. Buradan yola çıkarak, önce hem hatalı hem de hatasız durumda verilen tavrı çalıştırabilen bir spesifikasyonun gerekli ve uygun koşulları tanımlanmıştır. Daha sonra gösterilmiştir ki, tanımlanan koşulların ihlal edildiği durumlarda supremal bir arızaya dayanıklı alt dil belirlenmesi mümkündür. Son olarak bu alt dilin bulunabilmesini sağlayan bir algoritma geliştirilmiş ve bunun doğruluğu kanıtlanmıştır. Var olan diğer

çalışmalardan farklı olarak, bizim geliştirdiğimiz hataya dayanıklı kontrolör, hatanın oluşmasına ve sistemin herhangi bir zamanda tamirine izin vermektedir.

Hata kurtarıcı kontrole ilişkin, ayrık olaylı sistemlerin operasyonunda hem hata sonrası durumuna hem de tamir sonrası durumunun kontrolü için çalışılmıştır. Öncelikle DES'e yönelik arıza kurtarma için yeni bir model geliştirilmiştir. Özellikle, bir arızadan sonra bozuk bir spesifikasyona göre işlemine devam eden ve sonunda da arızadan sonra istenen bir davranışa yönelen bir arıza oluşumuna kadar, belirtilen nominal sistem davranışını takip eden bir arıza kurtarma denetçisi hesapladık. Ardından, yöntemimizin sistem tamirine de uygulanabileceği gösterilmiş ve isteğe bağlı sayıda arıza oluşumu ve sistem tamiri durumunu kontrol edebilebilen bir denetleyiçi oluşturulmasını sağlayan yinelemeli bir algoritma önerilmiştir.

Son olarak, arıza kurtarma ve tamir yöntemi, birçok ve farklı arıza ve tamirler ile birlikte sunulmuştur. Sonuç olarak, arızasız durumlarda belirtilen nominal sistem davranışını takip eden, her bir arıza tipi için istenen bozuk davranışa yönelen ve ilgili tamir işleminden sonra nominal davranışı kurtaran bir denetçi elde edilmiştir. Geliştirilen tüm yöntemler ayrı ayrı oluşturduğumuz bir küçük üretim sistemi örneğiyle gösterilmiş ve tüm sonuçlar aynı örnek üzerinde açıklanmıştır.

Keywords: Ayrık Olaylı Sistemler, Denetleyici Kontrol, Hata, Hata Dayanıklılığı, Hata Kurtarma, Tamir.

ACKNOWLEDGEMENTS

I would like to express my gratitude towards my thesis advisor Assoc. Prof. Dr. Klaus Werner SCHMIDT, for guiding me in all terms for my research. It was a great opportunity and honor to work with him during the preparation process of my thesis. I really would like to thank him for sharing any knowledge I needed with me and supporting me with this best all the time during my study.

I would like to express my gratitude, love and respect to my family and my husband for encouraging and supporting me all the time.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

FIGURES

LIST OF ABBREVIATIONS

DES	Discrete Event Systems
G	Plant Model Automaton
S	Supervisor
S^{FT}	Fault Tolerant Supervisor
C	Specification Automaton
K	Specification
KN	Nominal Specification
KD	Degraded Spefication
KF	Faulty Specification
KR	Specification Under Repair
$\sum$	Finite Set of Events
X	Finite Set of States
F	Faulty Event
r	Repair Event
δ	A Partial Transition Function
x_0	The Initial State
X_m	The Marked States
L_m	Marked Language
$\sum_u$	Uncontrollable Event Set
$\sum_f$	Faulty Event Set
M_1	First Machine
M_2	Second Machine
s	String
ε	Empty String
p	Natural Projection Operation

CHAPTER 1

INTRODUCTION

Discrete Event Systems (DES) are introduced to model dynamic systems with a discrete state space. DES are characterized by *'states'*, *'events'*, whereby *'transitions'* between system states occur instantaneously with the occurrence of events. Faults are uncontrollable events that suddenly happen in a system and have a negative effect on the behaviour of the DES. To deal with faults, the first problem to be addressed is fault diagnosis. Here, diagnosers are used to understand if and where a fault occurs [1]. Moreover, fault recovery or fault tolerant control enable operating a system under potentially relaxed performance specifications, even in case of faults. In this context, there is a distinction between passive and active approaches to fault tolerance in the literature [2]. Passive fault tolerance allows using the same controller in both faulty and non-faulty cases. Active fault-tolerance requires a controller adapting its control law, in case a fault arouses. Such controller (either explicitly or implicitly) is based on a fault detection unit in order to make an adjustment on its operation. That is, fault-recovery control allows maintaining the system operation after an event of fault, while performing a potentially degraded specification.

The subject of this thesis is the fault-tolerant and fault-recovery control of DES. Hereby, the thesis has three main contributions:

1. Computation of Fault-Tolerant Supervisors for Discrete Event Systems: The main idea of this contribution is creating a single supervisor, that makes it possible to realize a supremal fault-tolerant sublanguage for operating the system both in the nominal and in the faulty case. This supervisor also handles system repair in the sense of returning to the nominal system behavior in case the fault is repaired.

2. Computation of Supervisors for Fault-Recovery and Repair for Discrete Event Systems: For this contribution, we propose a new method for the fault-recovery of DES. The method computes a fault-recovery supervisor that follows the specified nominal system behavior until a fault-occurrence, that continues its operation according to a degraded specification after a fault and that finally converges to a desired behavior after fault. Also our method is extended to include system repair such that the system returns to its nominal behavior.

3. Discrete Event Supervisor Design and Application for Manufacturing Systems with Arbitrary Faults and Repairs: The third contribution extends the method in second contribution. With the extended method, our fault-recovery supervisor enables operating a DES with multiple different faults.

Considering the first contribution, we center upon the problem of passive fault tolerance for discrete event systems (DES). We address faults which disable the operation of some components belonging to the DES plant. In that, our fault model suggests that a certain set of plant events is no longer possible in case a fault happens. In turn, it is possible for the plant to return to its nominal operation after repair. In the described setting, we investigate the existence and synthesis of a fault-tolerant supervisor which accomplishes the desired closed-loop behavior in both non-faulty and faulty cases. To this end, our first contribution is the derivation of the sufficient and necessary conditions for the existence of such fault-tolerant supervisor, accompanied by a polynomial-time verification algorithm. In case these conditions are not achieved, the determination of fault-tolerant sublanguages of a given specification is required. The second contribution we carried out is we show that a supremal fault-tolerant sublanguage exists and we develop a polynomial-time algorithm for its computation. The main ideas of this contribution are published in the conference paper [3].

Considering the second contribution, we develop a new approach for fault-recovery of DES, which is also suitable for system repair. We discuss our DES model including the occurrence of faults, and we use three language specifications to represent the desired system behavior in a convenient manner: the desired non-faulty behavior is

provided by a nominal specification; the desired continuation of the system behavior after a fault-occurrence is provided by a degraded specification; and the desired faulty closed-loop behavior which is required to be achieved finally is represented by a faulty specification that is more restrictive. As a crucial feature of our formulation, it is not assumed that it is required for the closed-loop system to obey each specification starting from the initial plant state, but it has to achieve each specification partially, depending on the presence of a fault. We propose an algorithm for finding a non-blocking fault-recovery supervisor which is based on the interleaving composition operation [4] and uses language convergence [5], in order to solve the fault-recovery problem. We further show that it is also possible to apply the developed method to handle system repair. Then, it is desired to achieve the nominal specification at the end, after a system repair is performed. Finally, an iterative application of our method allows computing a fault-recovery supervisor for an arbitrary number of occurrences of fault and system repairs. The main ideas of this contribution are published in the conference paper [6].

Considering the third contribution, we extend our fault-recovery control method to the case of different faults which are possible to occur in an arbitrary order. To that end, at first we develop a general formulation of the problem setting, and then suggest a new algorithm for the computation of a fault-recovery and repair supervisor. The main idea of the algorithm is to compute supervisors iteratively, which can handle an increasing number of successive faults and repairs, and verify whether a new behavior is observed in each of the iterations of the algorithm or not. Once no new behavior is observed, the algorithm terminates with the solution supervisor. We apply the fault-tolerant method, fault-recovery method, and developed method to a small manufacturing system example for the purpose of illustration. The main ideas of this contribution are published in the conference paper [7].

There are different approaches developed for the fault-tolerant control of DES in the existing literature, under different assumptions. In [8], if a fault occurs, the system follows a transient mode, and then if the fault is detected, it enters a recovery mode. It is assumed that it is possible to detect faults within a known bounded delay of

event occurrences. In this setting, the paper determines a supervisor which performs design specifications in different operation modes. Unlike our approach, the proposed method requires the closed-loop system to obey each specification, starting from the initial plant state. Nevertheless, the case of system repair and re-occurrence of faults are not incorporated by [8].

The work by [9, 10] suggests to detect faults and then to switch to a different supervisor, before the nominal system behavior is violated. This approach is based on using a diagnoser to detect faults and requires the formulation of a modified system specification for each system state, where it is possible to detect a fault. Unlike our fault-tolerant approach, the supervisor in [10] switches to a different operation following each separate fault detections. In addition, system repair is not included and the re-occurrence of faults is not addressed.

[11] studies conditions for the control-reconfiguration in case of faults. After the fault occurrence, a post-fault specification has to be performed, by means of which controller reconfiguration is only possible following fault detection. The paper provides sufficient and necessary conditions for the existence of controller in this setting. It also determines the additional conditions under which no control reconfiguration is required. However, unlike our approach, [11] is only concerned with supervisor existence, rather than synthesis, and does not support non-blocking supervision. Furthermore, system repair and re-occurrence of faults are not addressed.

[12] proposes the use of fault-accommodating models. With this approach, integrating the nominal and faulty system behavior and system specification into a single model is possible. Then, a classical supervisor synthesis problem can be solved under partial observation in order to achieve fault-tolerance. There is no need for an explicit switching mechanism to implement the designed supervisor. However, unlike our approach, the paper requires the formulation of a fault-accommodating specification which should be met starting from the initial plant state. Moreover, repair and re-occurrence of faults are not addressed.

A further line of work studies on the robust supervisory control of DES [13, 14,

15, 16, 17]. In this setting, different plant models are used to represent potential plant behaviors, such as under fault. Unlike our method, it is assumed that one of the models is active (depending on which fault happens), but it is not captured that a fault might switch the system behavior from one model to another model. A certain application of robust control to fault-tolerance presented by [18] is based on the identification of tolerable fault sequences. However, that paper only addresses existence conditions and does not carry out supervisor design.

The idea of convergence is adopted for fault-tolerance in terms of recovering the nominal system behavior after a fault in [19]. [19] defines fault-tolerance from the point of view that the system behavior should converge to the nominal system behavior after a finite number of event occurrences upon the occurrence of any fault. The paper provides necessary and sufficient conditions for the supervisor existence in this setting. In addition, [20] proposes a method for the computation of fault-tolerant supervisors. Unlike our approach, [19, 20] require that a fault must be reversible after a bounded delay, whereby it is required for the closed-loop system to obey a language specification starting from the initial plant state. Moreover, system repair and the re-occurrence of faults are not addressed.

In the recent literature, [21] considers the relaxation of the nominal specification in order to avoid restricting the system behavior unnecessarily. It is suggested to allow potentially faulty behavior and handle undesired behavior on an upper level of the control architecture. Similar to the other related literature, system repair and the re-occurrence of faults are not addressed in that paper.

The outline of this thesis is as follows. Chapter 2 provides the necessary background on DES and the supervisory control of DES. In Chapter 3, our new method for the computation of fault-tolerant supervisors including problem statement, problem solution and solution algorithm is presented. Our new approach for the computation of supervisors for fault-recovery and repair is addressed in Chapter 4 and extended to the case of multiple different faults in Chapter 5. Chapter 6 gives conclusions and points out directions for future work.

CHAPTER 2

BACKGROUND

2.1 Discrete Event Systems

Discrete Event Systems (DES) represent dynamic systems with a discrete state space. Here, *states* represent the passing of time and state *transitions* happen instantaneously based on the ocurrence of discrete *events*. Various human-made systems can be modeled by DES. DES are for example used in computer systems, manufacturing systems, communication systems, etc. [1].

A small example for DES modeling is a simple fan sytem. The simple fan system has three discrete states: `OFF`, `LOW`, `HIGH` and three discrete events: `stop`, `startLow`, `changeHigh`. Initially the fan state is `OFF`. In this state, the event `startLow` can happen and creates a transition to the state `LOW`. In the `LOW` state, the fan turns slowly and works in this situation until a new event happens. In this state there are two possible events: `stop` and `changeHigh`. If `stop` happens, the system transitions to `OFF` again. Otherwise, the system goes to the state `HIGH` and the fan starts to work faster. If `stop` happens in the state `HIGH`, the system goes to the state `OFF` again.

2.2 Formal Language

A language is describing the logical behavior of DES. An alphabet is defined as a finite set of events and it is denoted by Σ. Every finite event sequence from Σ is called as string s. If a string does not have any event, it is an empty string and denoted by ε. $|s|$ defines the length of a string s. It means $|s|$ gives the number of events in a string. So the length of an empty string is zero, $|\varepsilon| = 0$. A language is defined as a set of

finite-length strings from events in Σ [1].

For example, the alphabet of the fan is $\Sigma = \{\texttt{stop}, \texttt{startLow}, \texttt{changeHigh}\}$. Some example strings are: $s_1 = \texttt{startLow stop}$, $s_2 = \texttt{startLow changeHigh stop startLow}$ etc. And the length of these strings are $|s_1| = 2$ and $|s_2| = 4$. We can define a language for the system like:

$L = \{\varepsilon, \texttt{startLow}, \texttt{startLow changeHigh stop}\}$.

The set of all finite strings of elements of Σ is called the *Kleene-closure* and denoted by Σ^*. Then we can say that L is a *subset* of Σ^*. As an example, *Kleene-closure* of the simple fan system is

$$\begin{aligned}\Sigma^* = \{&\varepsilon, \texttt{startLow}, \texttt{changeHigh}, \texttt{stop}, \texttt{startLow stop},\\ &\texttt{startLow stop startLow startLow stop changeHigh},\\ &\texttt{startLow stop changeHigh stop}, \ldots\}\end{aligned} \tag{2.1}$$

The *concatenation* and further string properties are explained as follows:

- Let two strings $s1, s2 \in \Sigma^*$. The *concatenation* of these two strings is $s = s1\,s2$.
- $s1$ is called the *prefix* of string s.
- $s2$ is called the *suffix* of string s.

The *prefix-closure* is a language operation. The *prefix-closure* of a language $L \in \Sigma^*$ is denoted by $\overline{L}$ and it is defined as $\overline{L} := \{s_1 \in \Sigma^* | \exists s \in L \text{ s.t. } s_1 \leq s\}$. If $L = \overline{L}$, the language L is called as *prefix closed*.

Another language operation is the *natural projection*. Let $\hat{\Sigma} \subseteq \Sigma$. The natural projection erases all events in Σ that do not belong to a defined subset $\hat{\Sigma}$ of Σ. This operation is written as $p : \Sigma^* \to \hat{\Sigma}^*$. Assume that $s \in \Sigma^*$ and $\sigma \in \Sigma$. The natural

projection is defined such that:

$$\begin{aligned} p(\varepsilon) &= \varepsilon \\ p(\sigma) &= \begin{cases} \sigma & \text{if } \sigma \in \hat{\Sigma} \\ \varepsilon & \text{otherwise} \end{cases} \\ p(s\sigma) &= p(s)p(\sigma). \end{aligned} \tag{2.2}$$

The inverse projection is denoted as $p^{-1} : \Sigma^* \to 2^{\Sigma^*}$. It is defined such that for each $t \in \Sigma^* : p^{-1}(t) = \{s \in \Sigma^* p(s) = t\}$.

2.3 Automata

An automaton is used to represent a language and model discrete event systems. An automaton is denoted by $G = (X, \Sigma, \delta, x_0, X_m)$. G is a fivetuple such that:

- X: finite set of states
- Σ: a finite set of events
- δ: a partial transition function
- $x_0 \in \Sigma$: the initial state
- $X_m \subseteq X$: the marked states (desired states)

The connection between languages and automata is created by the state transition diagram of an automaton. For the state transition diagram of an automaton; circles represent states and the diagram starts with the initial state, arrows represent transitions between states and arrows are named by corresponding events. An example of the state transition diagram for a simple fan system is shown in figure 1.

The *closed language* $L(G)$ is $L(G) := \{s \in \Sigma^* | \exists \delta(x_0, s)!\}$. The *closed language* $L(G)$ contains all possible event sequences starting from initial state of G to each

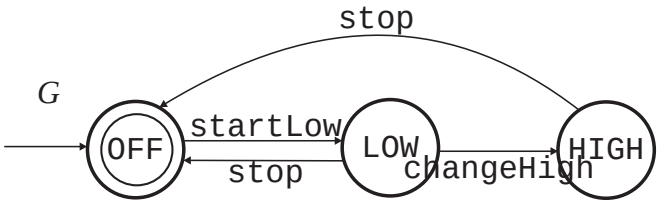


Figure 1: State transition diagram for the simple fan system.

states. The *marked language* $L_m(G)$ is $L_m(G) := \{s \in L(G) | \exists \delta(x_0, s) \subset X_m\}$. The *marked language* $L_m(G)$ contains all possible event sequences starting from initial state of G to a marked state.

If $\overline{L_m(G)} = L(G)$, then the finite state automaton is called as *nonblocking*. A non-blocking automaton means that there is a path from every state of G to a marked (desired) state. If $\overline{L_m(G)} \subset L(G)$, G is blocking. If a string in an automaton G starts from a state and after some transition goes back to the same state, then the automaton G has a cycle. If any automaton does not have any cycle, it is called as acyclic.

Let $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$ and $G' = (X', \Sigma, \delta', x_0', X_\mathrm{m}')$ be finite state automata. G' is a *subautomaton* of G, denoted as $G' \sqsubseteq G$ if either G' is the empty automaton ($X' = \emptyset$), or $X' \subseteq X$, and for all $x \in X'$ and $\sigma \in \Sigma$, it holds that $\delta'(x, \sigma)! \Rightarrow \delta'(x, \sigma) = \delta(x, \sigma)$.

There is some important automata operations to analyze any DES or modify state space diagrams:

- *Accessible*: If all states in X are reachable from the initial state x_0, then the automaton G is *accessible*:

  $\forall x \in X, \exists s \in \Sigma^\star \text{such that} \delta(x_0, s) = x$

  The operation $Acc(G)$ makes G accessible by removing all non-accessible states from X.

- *Coaccessible*: If each state in X reach to a marked state, the automaton G is *coaccessible*:

  $\forall x \in X, \exists s \in \Sigma^\star$ such that $\delta(x, s) \in X_m$

Now, we can say that, if an automaton is coaccessible, it is directly nonblocking. The operation $CoAcc(G)$ makes G coaccessible by removing all no marked state reachable states from X.

- *Trim*: If an automaton G is both accessible and coaccessible, G is trim.

$$Trim(G) := CoAcc[Acc(G)] := Acc[CoAcc(G)] \tag{2.3}$$

- *Synchronous Composition*: This operation allows to synchronize two different automata. It makes it possible to model one system by more than one automaton. When sychronous composition operation is applied to two or more automata, the output will become a bigger automaton that captures the joint behavior of both automata. Now assume $G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1}, X_{\mathrm{m},1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2}, X_{\mathrm{m},2})$ are two different automata and the synchronous composition of these two automata is written as:

$$G_1 || G_2 = G_{12} = (X_{12}, \Sigma_{12}, \delta_{12}, x_{0,12}, X_{\mathrm{m},12}) \tag{2.4}$$

Synchronous composition operation states are $X_{12} = X_1 \times X_2$ (*the canonical product of states from* X_1 *and* X_2), the events are $\Sigma_{12} = \Sigma_1 \cup \Sigma_2$ (*the union of events in* Σ_1 *and* Σ_2), the initial state is $x_{0,12} = (x_{0,1}, x_{0,2})$, the marked states are $X_{\mathrm{m},12} = X_{\mathrm{m},1} \times X_{\mathrm{m},2}$. The transition makes sure that the events in $\Sigma_1 \cap \Sigma_2$ that are shared by G_1 and G_2 are synchronized. For $(x_1, x_2) \in X_{12}$ and $\sigma \in \Sigma_{12}$:

$$\delta_{12}((x_1,x_2),\sigma) = \begin{cases} (\delta_1(x_1,\sigma), \delta_2(x_2,\sigma) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \wedge \delta_1(x_1,\sigma)! \wedge \delta_2(x_2,\sigma)! \\ (\delta_1(x_1,\sigma), x_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \wedge \delta_1(x_1,\sigma)! \\ (x_1, \delta_2(x_2,\sigma)) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \wedge \delta_2(x_2,\sigma)! \end{cases} \tag{2.5}$$

2.4 Supervisory Control

The supervisory control theory is introduced in [22]. The theory is based on creating a controller called as supervisor which allows or disables corresponding events to reach the desired behaviour called as specification.

Assume that $G = (X, \Sigma, \delta, x_0, X_m)$ is a given plant automaton. Σ_c is the *controllable event set* and Σ_u is the *uncontrollable event set*. Then the alphabet of the plant is defined as: $\Sigma = \Sigma_c \cup \Sigma_u$. Controllable events can be disabled by a supervisor, but uncontrollable events cannot.

The supervisor is denoted by S and it is also defined as a finite state automaton like: $S = (Q, \Sigma, \nu, q_0, Q_m)$. The closed loop behavior is given by the plant G controlled by the supevisor S and is computed as the synchronous composition of the plant $G||S$. The closed language of the closed loop is $L(G)||L(S)$ and the marked language of the closed loop is $L_m(G)||L_m(S)$. S is a nonblocking supervisor if $L(G)||L(S)$ is nonblocking. As said before, supervisor S cannot disable uncontrollable events in Σ_u. So, it must be true for all $\sigma \in \Sigma_u$ and $s \in L(G) \cap L(S)$, if $s\sigma \in L(G)$, $s\sigma \in L(S)$.

The specification represents the desired closed-loop behavior and it is also defined by an automaton: $C = (Y, \Sigma, \beta, y_0, Y_m)$. The specification language is denoted as $K = L_m(C)$. The specification contains all desired strings and the supervisor disables undesired strings according to the specification. The specification is called as *controllable* for $L(G)$ and Σ_u if it does not force the supervisor to disable the uncontrollable events. A controllable specification fulfills

$$\overline{K} = \Sigma_u \cap L(G) \subseteq \overline{K} \tag{2.6}$$

If the specification K is not controllable for $L(G)$ and Σ_u, the supervisor should implement the *supremal controllable sublanguage* of K. There is a *SupCon* algorithm which is used to find the supremal controllable largest possible sublanguge $K_{sub} \in K$.

Such that:

$$L_m(S||G) = SupC(K, L(G), \Sigma_u) \tag{2.7}$$

2.5 Interleaving Composition

We recall the *interleaving composition* from [4]. Given two languages $K_1, K_2 \subseteq \Sigma^\star$ over the same alphabet, it defines a language that contains all possible interleavings of strings from K_1 and K_2. We reformulate the interleaving composition in our notation.

Definition 1. *Let Σ be an alphabet and $K_1, K_2 \subseteq \Sigma^\star$ be two languages. The* interleaving composition *$K_1|||K_2$ of K_1 and K_2 is defined such that*

$$s \in K_1|||K_2 \Leftrightarrow s = s_1^1 s_1^2 \cdots s_k^1 s_k^2 \text{ for some } k \in \mathbb{N} \text{ and } s_1^j s_2^j \cdots s_k^j \in K_j \text{ for } j = 1, 2.$$

2.6 Language Convergence

We employ the notion of *language convergence* as introduced by [5]. For a string $s \in \Sigma^\star$, we write $suf_i(s)$ for the string obtained by deleting the first i events from s. Specifically, $suf_0(s) = s$ and $suf_{|s|}(s) = \varepsilon$. Then, for a language $L \subseteq \Sigma^\star$, the *suffix closure* of L is the subset of all suffixes of strings in L:

$$suf(L) = \{suf_i(s) | s \in L, i \leq |s|\}.$$

A language is suffix-closed if $suf(L) = L$. Now consider two languages $M, K \subseteq \Sigma^*$. M is said to *converge* to K, denoted by $K \Leftarrow M$, if there is an integer $n \in \mathbb{N}_0$ such that for each $s \in M$, there exists an $i \leq n$ such that $\text{suf}_i(s) \in K$. The least possible n is called the *convergence time*.

In the supervisory control context, the *controlled convergence problem* (CCP) is studied. Let G be a plant automaton over the alphabet Σ, $\Sigma_u \subseteq \Sigma$ be a set of uncon-

trollable events and $K \subseteq \Sigma^*$ specification. A supervisor S is said to solve the CCP for G, K and Σ_u if $S||G$ is nonblocking, $L(S||G)$ is controllable for $L(G)$ and Σ_u, and $K \Leftarrow L_m(S||G)$. Assume that X is the state set of G and Y is the state set of a recognizer C such that $L_m(C) = K$. It is shown by [5] that the solvability of the CCP can be decided by an algorithm with complexity $\mathcal{O}(|X|^2 2^{2|Y|})$. Whereby it is noted that the synthesis neither offers a unique nor an optimal solution.

We further extend the notion of language convergence to language convergence after a given language.

Definition 2. *Consider languages $M, K, L \subseteq \Sigma^*$, whereby $L \subseteq \overline{M}$. M is said to* converge *to K after L if $K \Leftarrow M/L$.*

Furthermore, a supervisor S is said to achieve language convergence for K after L if

$$K \Leftarrow (L_m(S)||M)/L.$$

An algorithm for the computation of such supervisor is adapted from [5] and implemented in [23]. It runs with the same complexity as the computation of a supervisor for the CCP. We denote the resulting language as $CA(K, M, L, \Sigma_u)$.

2.7 Motivating Example

The main subject of this thesis is fault-tolerant and fault-recovery control of DES. In this section, we provide a motivating example which is used to explain our study topics. This is a simple manufacturing system and it has two simple machines and shown in Fig. 2.

.

The machines are denoted as M_1 for the first machine and M_2 for the second machine. Products can enter M_1 and M_2 with the events `in1` and `in2`, respectively. The first machine M_1 can process products with event `op1` and the second machine

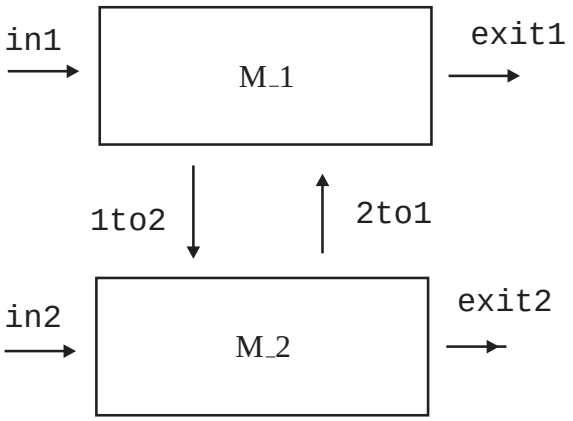


Figure 2: Schematic of the example system.

M_2 can process products with event `op2`. Also there is a connection between M_1 and M_2 such that products can travel between these two machines with the events `1to2` and `2to1`. There are two different exit points for products: one of them is from M_1 with event `exit1` and the other one is from M_2 with event `exit2`. For a nominal behavior of the system, all the defined events can happen. System works as the products can enter the system to first or second machine and operated by them before leaving the system.

While the system is working in a nominal behavior, faults can occur in the system. When a fault occurs, it is for example the case that an event cannot happen anymore. Figure 3 shows the example system with a fault.

For the motivating example we define an example faulty behavior. When the fault occurs, *fault* event happens. For example, we assume that fault occurs on the first machine. Then `op1` event disappears and cannot happen anymore. Now the system has to run in the faulty behavior instead of the nominal behavior and a fault-recovery supervisor should take care that nothing undesired happens. Assume that fault happens when a product is on the first machine. The faulty-recovery behavior can be defined in three different ways. In the first case, the product waits on the first machine until the fault is repaired and the second machine continues its nominal behavior without any transition events (`1to2` and `2to1`) between first machine. In the second csae,

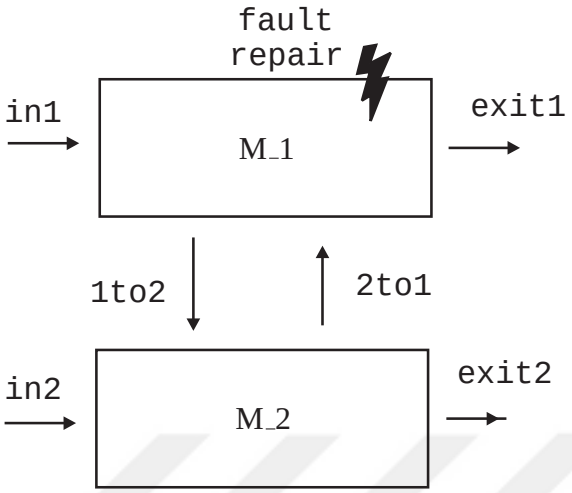


Figure 3: Schematic of the example system with fault events.

M_1 can send the unprocessed product to the outside with event `exit1` and does not take any more products until the fault is repaired. The system continues to work with M_2 in its nominal behavior. In the third case, the product is sent from M_1 to M_2 and processed there. After sending the product, M_1 does not take any more products and M_2 continues to work in the nominal bahavior. The first task of this thesis is to formally design fault-recovery supervisors that realize the appropriate behavior after a fault.

The *repair* event happens when the fault is repaired. The *repair* event causes that the behavior that was disabled by the fault is again possible. This means for the example that M_1 can restart processing products by the event *op1* and the system has to resume its nominal behavior after repair. In the first case discussed above, this means that the unprocessed product waiting in M_1 must be processed and M_1 can take new products again. In the other cases, this means that M_1 can take and process products again and no products need to be sent to M_2. The second task of this thesis is to design fault-recovery and repair superisors that correctly resume the nominal system operation after repair.

CHAPTER 3

COMPUTATION OF FAULT-TOLERANT SUPERVISORS FOR DISCRETE EVENT SYSTEMS

The first objective of this chapter is determining if a given supervisor can tolerate a given set of faulty events. The second objective is computing fault-tolerant supervisors that can tolerate a given set of faulty events. The section outline of this chapter is as follows. Section 3.1 presents a motivating example. Section 3.2 formulizes the problem statement. Section 3.3 considers the verification of fault-tolerance. Section 3.4 establishes the existence of a supremal fault-tolerant sublanguage. Section 3.5 proposes an algorithm for the computation of this supremal fault-tolerant sublanguage. We note that the results presented in this chapter are published in the conference paper [3].

3.1 Motivation

We reconsider the example system in Section 2.7 and a model of the system is shown in Fig. 4.

We first determine component models for the two machines in order to create the plant model automaton G of the overall system. These models G_1 and G_2 are shown in Fig. 5. The parallel composition of G_1 and G_2 results in the overall system model $G = G_1||G_2$ that is also shown in Fig. 5.

The desired operation of our example system is as follows: If a product is processed by one of the machines then it should not be processed by the other machine. In addition, a product should be processed only one time and should not cycle be-

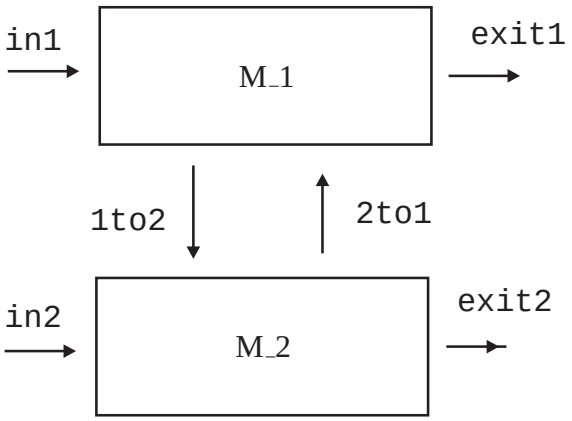


Figure 4: Schematic of the example system

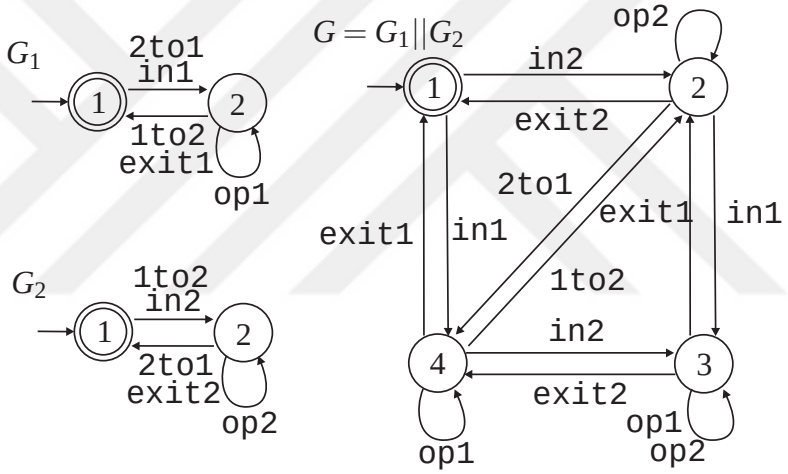


Figure 5: Plant model automata of the example system.

tween the machines. For example, a product can enter the system with `in1` and `op1` can happen afterwards. Then, the product will not be processed by `op2`. On the other hand, if the product enters the system with event `in1` and is not processed by `op1`, `1to2` and `op2` must happen. Fig. 6 shows the four component automata of our specification. The overall specification is $K = L_m(C_1||C_2||C_3||C_4)$.

The maximally permissive supervisor S such that $L_{\mathrm{m}}(G||S) = SupCon(L_{\mathrm{m}}(C), G, \Sigma_{\mathrm{u}})$ for this specification and plant is shown in Fig. 7. Here, we assume that $\Sigma_{\mathrm{u}} = \emptyset$.

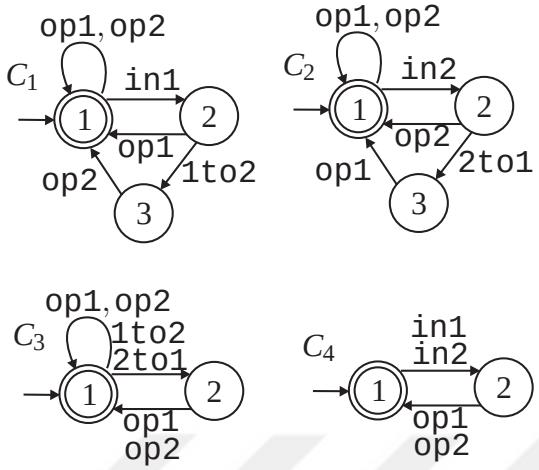


Figure 6: Components of the specification for the example system.

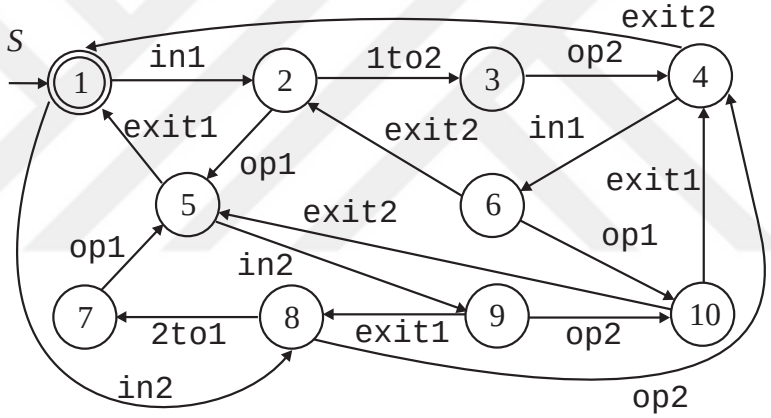


Figure 7: Maximally permissive supervisor *S* for the example system.

The supervisor *S* is designed to control the plant in the nominal case, that is, when the plant does not exhibit any faulty behavior. Now assume that faults occur in the plant. Such fault can for example be caused by the failure of a transport or processing unit. Consider the transport from M_1 to M_2 fails, that is, the event `1to2` cannot happen any more. In that case, the closed-loop behavior of *G* under supervision by *S* is represented by the automaton S^{1to2} in Fig. 8. It can be seen that, even in the case of a fault, the closed loop is nonblocking and fulfills the specification *K*. For example, if a fault happens when *S* in Fig. 7 is at state 3, the closed loop after the fault can continue its operation from that state but will not be able to return to that state unless the fault is repaired. That is, *S* can tolerate the faulty event `1to2`.

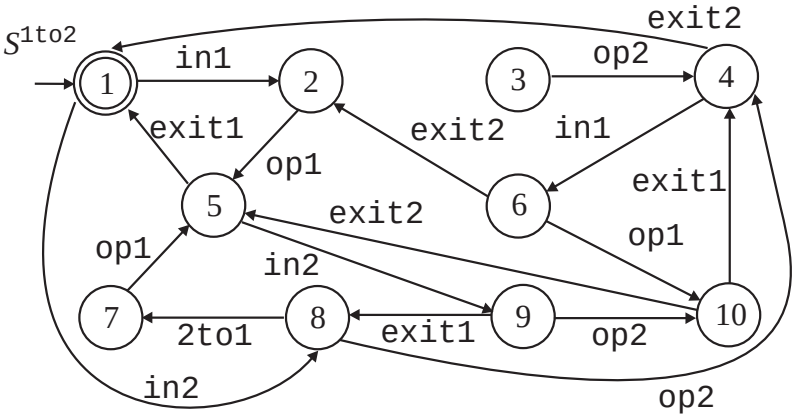


Figure 8: Closed loop with faulty event `AtoB`.

In contrast, assume that M_1 is faulty such that event `op1` is no longer possible. Then, the corresponding closed-loop behavior is given by the automaton S^{op1} in Fig. 9. It can be seen that the faulty closed loop becomes blocking. Hence, S cannot tolerate a fault in the event `op1`.

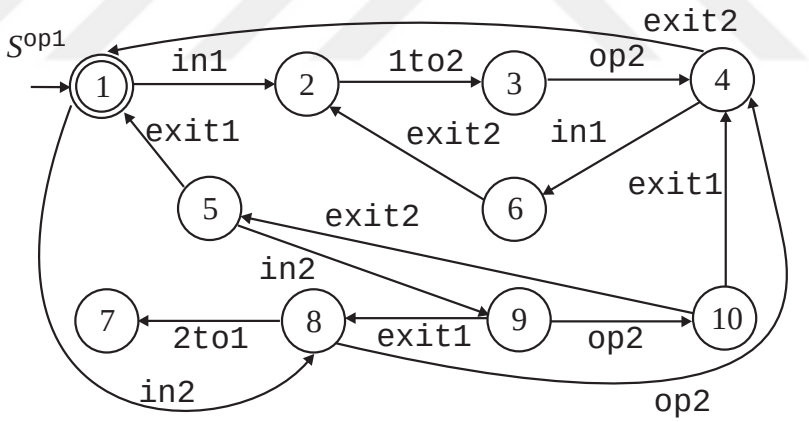


Figure 9: Closed loop with faulty event `op1`.

In view of the previous discussion, the first objective of this chapter is determining if a given supervisor can tolerate a given set of faulty events. The second objective is computing fault-tolerant supervisors that can tolerate a given set of faulty events.

3.2 Problem Formulation

According to the previous discussion, it is desired to find a supervisor that still fulfills the specified behavior even if a fault happens. We formalize this requirement by defining a fault-tolerant supervisor. We assume that $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$ is a plant automaton, $K \subseteq L_\mathrm{m}(G)$ is a specification and Σ_u is a set of uncontrollable events. In addition, we introduce the set of faulty events Σ_f that represents all events that can no longer occur if a fault happens. Finally, we introduce the faulty plant $G^\mathrm{f} = (X, \Sigma, \delta^\mathrm{f}, x_0, X_\mathrm{m}) \sqsubseteq G$ such that G^f is obtained from G by removing each transition with an event in Σ_f. For each $x \in X$, we write $G_x^\mathrm{f} = (X, \Sigma, \delta^\mathrm{f}, x, X_\mathrm{m})$ with the initial state x. We denote the maximally permissive supervisor for the given plant G and specification K as $S = (Q, \Sigma, v, q_0, Q_m)$ and write $S^{FT} = (Q^{FT}, \Sigma, v^{FT}, q_0^{FT}, Q_m^{FT})$ for the fault-tolerant supervisor.

For the given example in Section 3.1, the plant is $G = G_1||G_1$ in Fig. 5 and the set of uncontrollable events is $\Sigma_\mathrm{u} = \emptyset$. The maximally permissive supervisor S in Fig. 7 can be used to mark the specification $K = L_\mathrm{m}(S)$. The supervisor for the given plant and specification is S that is shown in Fig. 7. A fault tolerant supervisor which is nonblocking for faulty event $op1$ is S^{FT} that is shown in Fig. 10.

We now formalize our notion of fault-tolerance.

Definition 3. *Let $G, K, \Sigma_\mathrm{u}, \Sigma_\mathrm{f}$ be given as above. The specification K is fault-tolerant for $G, \Sigma_\mathrm{u}, \Sigma_\mathrm{f}$ if*

1. *K is controllable for G and Σ_u*
2. *for all $s \in \overline{K}$*

$$\overline{K/s \cap L_\mathrm{m}(G)/s \cap (\Sigma \setminus \Sigma_\mathrm{f})^\star} = \overline{K/s} \cap \overline{L_\mathrm{m}(G)/s} \cap (\Sigma \setminus \Sigma_\mathrm{f})^\star.$$

In words, a fault-tolerant specification has to be controllable for G and Σ_u according to (1) in Definition 3. In addition, it must be fulfilled for any string in K that,

whenever a fault happens, this means events in Σ_f are no longer possible, the string can still be completed to fulfill the specification.

For the example defined in Section 3.1 the supervisor S is blocking if the event $op1$ is a faulty event ($\Sigma_f = \{\texttt{op1}\}$). Fig. 9 shows that if `in2 2to1` happens from the initial state, S reaches state 7, from where there is no string to a marked state in S. Hence condition (2) in Definition 3 is violated and the specification $K = L_m(S)$ is not fault-tolerant. In contrast, the specification $K = L_m(S^{FT})$ for the automaton S^{FT} in Fig. 10 is fault-tolerant. It is controllable for G, Σ_u and S^{FT} is still nonblocking even if `op1` is no longer possible because of a fault. It is also readily observed that S^{FT} can be used as a fault-tolerant supervisor for $\Sigma_f = \{\texttt{op1}\}$.

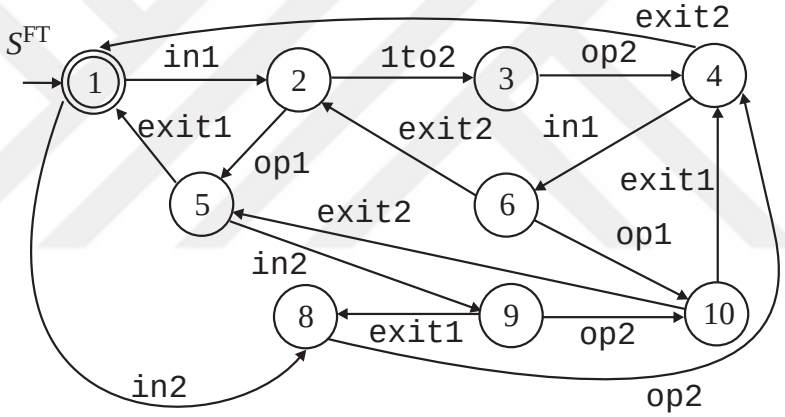


Figure 10: Example for a fault-tolerant supervisor.

Considering the previous discussion, it is desirable to determine a fault-tolerant specification according to Definition 3 and its associated supervisor whenever such specification exists. Hence, we intend to solve the following problem:

Problem 1. *Let G be a plant automaton, K be a specification, Σ_u be a set of uncontrollable events and Σ_f be a set of faulty events as introduced before. We want to find a supervisor S^{FT} for G and Σ_u such that:*

1. *$L_m(S^{FT}) \subseteq K$ is fault-tolerant for G, Σ_u, Σ_f*
2. *$L_m(G||S^{FT})$ is as large as possible.*

3.3 Verification of Fault Tolerance

Before solving Problem 1, we outline a procedure for the verification if a given specification K is fault-tolerant for a plant G and the alphabets Σ_u, Σ_f, that is, if the conditions in Definition 3 are fulfilled. Our result is stated in Lemma 1.

Lemma 1. *Let G, Σ_u, S and Σ_f be given and assume that $C = (Y, \Sigma, \gamma, y_0, Y_m)$ is a recognizer of K, that is, $L_m(C) = K$. Also define $R = G||C = (Z, \Sigma, \alpha, z_0, Z_m)$ and write $R^f = (Z, \Sigma, \alpha^f, z_0, Z_m)$ for the subautomaton of R where all transitions with events in Σ_f are removed. K is Σ_f-tolerant if and only if K is controllable for G and Σ_u and R^f is non-blocking.*

R^f represents the closed-loop system after a fault happened in the case that K is controllable for G and Σ_u. Lemma 1 states that it should be possible to reach a marked state in the faulty closed loop. Formally, Lemma 1 is proved as follows.

Proof. (IF) We assume that K is controllable for G and Σ_u and R^f is non-blocking. We have to show that the conditions in Definition 3 are fulfilled. (1) holds by assumption. Regarding (2), assume that $s \in \overline{K}$ and let $u' \in \overline{K/s} \cap \overline{L_m(G)/s} \cap (\Sigma \setminus \Sigma_f)^\star$. Write $x = \delta(x_0, s)$ and $y = \gamma(y_0, s)$. By definition of R, $s \in L(R)$ and $\alpha(z_0, s) = (x, y)$. Moreover, by definition of R^f, $\alpha^f((x,y), u')!$ since $u' \in (\Sigma \setminus \Sigma_f)^\star$. Since R^f is non-blocking and does not have any transition with events in Σ_f, there is a $u \in (\Sigma \setminus \Sigma_f)^\star$ such that $\alpha^f((x,y), u'u) \in Z_m$. Hence, $u'u \in K/s \cap L_m(G)/s \cap (\Sigma \setminus \Sigma_f)^\star$. Since u' was chosen arbitrarily, (2) in Definition 3 is fulfilled.

(ONLY IF) We assume that (1) and (2) in Definition 3 are fulfilled. We show that the conditions in Lemma 1 also hold. K is controllable for G and Σ_u by assumption. Now let $(x,y) \in Z$. Then there is an $s \in \Sigma^\star$ such that $\delta(x_0, s) = x$ and $\gamma(y_0, s) = y$. Because of (2) in Definition 3, there is a $u \in K/s \cap L_m(G)/s \cap (\Sigma \setminus \Sigma_f)^\star$. That is, $u \in L_m(C)/s$ and $u \in L_m(G)/s \cap (\Sigma \setminus \Sigma_f)^\star$, which implies that $\alpha^f((x,y), u) \in Z_m$. Hence, R^f is nonblocking. □

For our example with faulty event `1to2`, there is a string to a marked state from each state of $S^{\texttt{1to2}}$ in Fig. 8 that corresponds to the plant in Fig. 5 where transitions with event `1to2` are removed and the specification $K = L_{\mathrm{m}}(S)$ in Fig. 7. Hence, the supervisor S in Fig. 7 can be considered as fault-tolerant. However, if the faulty event is `op1`, the situation is different. Consider the faulty closed loop $S^{\texttt{op1}}$ in Fig. 9 for the specification $K = L_{\mathrm{m}}(S)$. It can be seen that there is no string to a marked state if state 7 is reached in $S^{\texttt{op1}}$ and fault-tolerance as in Definition 3 is violated.

We finally note that the complexity of the verification in Lemma 1 is determined by computations on the state space of $G||C$. Hence, we obtain $O(|X||Y|)$.

3.4 Supremal Fault-Tolerant Sublanguage

According to the definition, there might be different fault-tolerant supervisors for a given plant G, specification K, uncontrollable event set Σ_{u} and faulty event set Σ_{f}. Our second result shows that there is a unique supremal such supervisor that realizes the supremal fault-tolerant sublanguage of K. In order to establish this result, we first show that fault-tolerant sublanguages are closed under union.

Lemma 2. *Let G, K, Σ_{u} and Σ_{f} be as introduced before and assume that the two languages $K_1, K_2 \subseteq L_{\mathrm{m}}(G)$ are fault-tolerant for G, Σ_{u} and Σ_{f}. Then, $K := K_1 \cup K_2$ is also fault-tolerant for G, Σ_{u} and Σ_{f}.*

Proof. It has to be shown that K fulfills the conditions in Definition 3. Considering (1), it follows directly from the union-closure of controllable sublanguages that K is controllable for G and Σ_{u}.

For (2), we need to show that $\overline{K/s \cap L_{\mathrm{m}}(G)/s \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^\star} = \overline{K/s} \cap \overline{L_{\mathrm{m}}(G)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^\star$. It trivially holds that $\overline{K/s \cap L_{\mathrm{m}}(G)/s \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^\star} \subseteq \overline{K/s} \cap \overline{L_{\mathrm{m}}(G)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^\star$. In order to show that also $\overline{K/s \cap L_{\mathrm{m}}(G)/s \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^\star} \supseteq \overline{K/s} \cap \overline{L_{\mathrm{m}}(G)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^\star$, let

$u \in \overline{K/s} \cap \overline{L_{\mathrm{m}}(G)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star}$.

$$
\begin{aligned}
&\Rightarrow u \in \overline{K/s} = \overline{(K_1 \cup K_2)/s} \text{ and } u \in \overline{L_{\mathrm{m}}(G)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star} \\
&\Rightarrow u \in \overline{K_1/s} \cup \overline{K_2/s} \text{ and } u \in \overline{L_{\mathrm{m}}(G)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star} \\
&\Rightarrow u \in \overline{K_1/s} \cap \overline{L_{\mathrm{m}}(G)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star} \text{ and} \\
&\quad u \in \overline{K_2/s} \cap \overline{L_{\mathrm{m}}(G)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star} \\
&\Rightarrow u \in \overline{K_1/s \cap L_{\mathrm{m}}(G)/s \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star}} \text{ and} \\
&\quad u \in \overline{K_2/s \cap L_{\mathrm{m}}(G)/s \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star}} \\
&\Rightarrow u \in \overline{(K_1 \cup K_2)/s \cap L_{\mathrm{m}}(G)/s \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star}} \\
&\quad = \overline{(K)/s \cap L_{\mathrm{m}}(G)/s \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star}}.
\end{aligned}
$$

□

We next define the set $\mathscr{F}_G(K, \Sigma_{\mathrm{u}}, \Sigma_{\mathrm{f}})$ of all fault-tolerant sublanguages of a given specification K for G, Σ_{u}, Σ_{f}.

$$\mathscr{F}_G(K, \Sigma_{\mathrm{u}}, \Sigma_{\mathrm{f}}) = \{F \subseteq K | F \text{ is fault tolerant for } G, \Sigma_{\mathrm{u}}, \Sigma_{\mathrm{f}}\}.$$

Using $\mathscr{F}_G(K, \Sigma_{\mathrm{u}}, \Sigma_{\mathrm{f}})$ it is now possible to prove the existence of a supremal fault-tolerant sublanguage.

Theorem 1. *Let G, K, Σ_{u} and Σ_{f} be as introduced before. Then, there is a supremal element in $\mathscr{F}_G(K, \Sigma_{\mathrm{u}}, \Sigma_{\mathrm{f}})$ and is evaluated as*

$$SupFT(K, G, \Sigma_{\mathrm{f}}, \Sigma_{\mathrm{u}}) = \bigcup \{F | F \in \mathscr{F}_G(K, \Sigma_{\mathrm{u}}, \Sigma_{\mathrm{f}})\}.$$

Proof. The correctness of Theorem 1 directly follows from Lemma 2. □

3.5 Computation of SupConFT

Having established the existence of a supremal fault-tolerant sublanguage, we now study its computation for a given plant G, specification K with recognizer C, uncontrollable events Σ_u and faulty event set Σ_f. We propose the following algorithm for finding $SupFT(K, G, \Sigma_u, \Sigma_f)$.

Input: C, G, Σ_u, Σ_f
Procedure:

1. Determine the maximally permissive supervisor $S = (Q, \Sigma, \nu, q_0, Q_m)$ such that $L_m(S) = SupCon(K, G, \Sigma_u)$
2. Compute the subautomaton $S^f \sqsubseteq S$ by deleting all transitions with events in Σ_f from S.
3. Define the set $Q^f \subseteq Q$ as the set of states that are not coaccessible in S^f.
4. Compute the subautomaton $S' \sqsubseteq S$ by deleting all states in Q^f from S.
5. Compute an automaton $S^{FT} = (Q^{FT}, \Sigma, \nu^{FT}, q_0^{FT}, Q_m^{FT})$ such that $L_m(S^{FT}) = SupCon(L_m(S'), S, \Sigma_u)$.
6. Return the supremal fault-tolerant sublanguage $L_m(S^{FT})$.

Algorithm 1: Computation of $SupFT$.

This algorithm determines the supremal fault-tolerant sublanguage as follows. A set of bad states Q^f is identified from where it is not possible to reach a marked state in the maximally permissive closed loop. Any string that leads to such bad state must be disabled in order to achieve fault-tolerance according to Definition 3. Hence, we delete these bad states from the maximally permissive supervisor S and use the resulting subautomaton as specification for the computation of the supremal fault-tolerant sublanguage. The complexity of the algorithm is determined by the computation of supremal controllable sublanguages in step (1) and (5). Hence, we obtain $O(|X|^2 |Y|^2)$.

Theorem 2. *Assume S^{FT} is constructed according to Algorithm 1 and $L_m(S^{FT}) \neq \emptyset$. Then, $L_m(S^{FT}) = SupFT(K, G, \Sigma_u, \Sigma_f)$, that is, S^{FT} solves Problem 1.*

Proof. Is is clear that S^{FT} solves Problem 1 if $L_{\mathrm{m}}(S^{\mathrm{FT}}) = SupFT(K, G, \Sigma_{\mathrm{u}}, \Sigma_{\mathrm{f}}) \neq \emptyset$. Hence, it is only required to prove that $L_{\mathrm{m}}(S^{\mathrm{FT}}) = SupFT(K, G, \Sigma_{\mathrm{u}}, \Sigma_{\mathrm{f}})$. It has to be shown that (i) $L_{\mathrm{m}}(S^{\mathrm{FT}})$ is a fault-tolerant sublanguage of K for G, Σ_{u}, Σ_{f} and (ii) for any $K \subseteq K' \supseteq L_{\mathrm{m}}(S^{\mathrm{FT}})$, it holds that K' is not fault-tolerant for G, Σ_{u}, Σ_{f}.

Considering (i), it holds that $L_{\mathrm{m}}(S^{\mathrm{FT}}) \subseteq K$ and $L_{\mathrm{m}}(S^{\mathrm{FT}})$ is controllable for G and Σ_{u} because of (4) and (5) in Algorithm 1. In order to show (2) in Definition 3, let $s \in L(G||S^{\mathrm{FT}})$ and $u' \in \overline{L_{\mathrm{m}}(S^{\mathrm{FT}})/s \cap L_{\mathrm{m}}(G)/s \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star}}$. That is, $s \in \overline{L_{\mathrm{m}}(S^{\mathrm{FT}})}$ and $u' \in \overline{L_{\mathrm{m}}(S)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star}$ according to the computation of S^{FT} from S. Then, because of (3), (4), (5) in Algorithm 1, there must be a $u \in (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star}$ such that $u'u \in L_{\mathrm{m}}(S^{\mathrm{FT}})/s$. Since $L_{\mathrm{m}}(S^{\mathrm{FT}}) \subseteq L_{\mathrm{m}}(G)$, also $u'u \in L_{\mathrm{m}}(G)/s$. That is, we confirmed that $u' \in \overline{L_{\mathrm{m}}(S^{\mathrm{FT}})/s} \cap \overline{L_{\mathrm{m}}(G)/s} \cap (\Sigma \setminus \Sigma_{\mathrm{f}})^{\star}$.

In order to show that $L_{\mathrm{m}}(S^{\mathrm{FT}})$ is indeed supremal, we assume there is an automaton $\hat{S}$ such that $K \supseteq L_{\mathrm{m}}(\hat{S}) \supset L_{\mathrm{m}}(S^{\mathrm{FT}})$ and $L_{\mathrm{m}}(\hat{S})$ is fault-tolerant for G, Σ_{u}, Σ_{f}. Then, there must be a string $s \in L_{\mathrm{m}}(\hat{S}) \setminus L_{\mathrm{m}}(S^{\mathrm{FT}}) = L_{\mathrm{m}}(\hat{S}) \setminus SupCon(L_{\mathrm{m}}(S'), G, \Sigma_{\mathrm{u}})$. That is, there must be an extension $u \in \Sigma^{\star}$ such that $\nu(q_0, su) \in Q^{\mathrm{f}}$ leads to a bad state in S because of (4) and cannot be prevented from occurring because of (5) in Algorithm 1. Hence, $L_{\mathrm{m}}(\hat{S})$ cannot be fault-tolerant for G, Σ_{u}, Σ_{f}, which contradicts the assumption. Hence, $L_{\mathrm{m}}(S^{\mathrm{FT}})$ is the supremal fault-tolerant sublanguage. □

Regarding the example in Section 3.1, the automaton $S^{\texttt{op1}}$ in Fig. 9 corresponds to the automaton S^{f} in Algorithm 1. Hence, the bad state 7 has to be removed from S in Fig. 7. The resulting automaton is controllable for G and Σ_{u} such that $S^{\mathrm{FT}} = S'$ is obtained in Fig. 10. This supervisor leads to a non-blocking closed-loop both in the nominal case and in the case of a faulty event `op1`. It also has to be noted that the closed loop returns to its nominal operation, whenever the fault is repaired. Hereby, a repair is possible any time since $L_{\mathrm{m}}(S^{\mathrm{FT}}) \subseteq L_{\mathrm{m}}(S)$.

CHAPTER 4

COMPUTATION OF SUPERVISORS FOR FAULT-RECOVERY AND REPAIR FOR DISCRETE EVENT SYSTEMS

In this chapter, a new method is developed for the fault-recovery of DES. According to this method, a fault-recovery supervisor is given that follows the specified *nominal* system behavior until a fault-occurrence, that continues its operation according to a *degraded* specification after a fault and that finally converges to a desired *behavior after fault*. This method can also be applied to system repair and we propose an iterative procedure that determines a supervisor for an arbitrary number of fault occurrences and system repairs.

The outline of this chapter is as follows. Section 4.1 formulizes the fault-recovery problem. Section 4.2 constructs a specification language and proposes a modified version of the language convergence problem. Section 4.3 shows to apply the method on an example. Section 4.4 proposes the method to system repair. Section 4.5 shows that the method satisfies the case of multiple fault occurrences and repairs. We note that the results presented in this chapter are published in the conference paper [6].

4.1 Problem Statement

In this section, we formulate the fault-recovery problem studied in this paper. We consider that the system is modeled using the alphabets $\Sigma, \Sigma^{\mathrm{N}}, \Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}}$. Hereby, Σ^{F} contains fault events whose occurrence indicates the occurrence of a fault, Σ^{N} contains all events that are not associated to faults and $\Sigma = \Sigma^{\mathrm{N}} \dot{\cup} \Sigma^{\mathrm{F}}$. Σ_{u} is the set of uncontrollable events. Then, the system behavior is characterized by the plant model $G = (X, \Sigma, \delta, x_0, X_{\mathrm{m}})$ that includes the potentially faulty system behavior.

The main objective is to synthesize a supervisor $S^{\mathrm{F}} = (Q^{\mathrm{F}}, \Sigma, \nu^{\mathrm{F}}, q_0^{\mathrm{F}}, Q_{\mathrm{m}}^{\mathrm{F}})$ that achieves fault-recovery in the closed loop $G||S^{\mathrm{F}}$. In order to specify the desired system behavior, we consider three different specifications. First, the nominal specification $K^N \subseteq L_{\mathrm{m}}(G)$ characterizes the desired system behavior in case no fault is present in the system. That is, the closed-loop behavior without any fault occurrence should be a subset of the nominal specification and nonblocking as stated in the following condition.

P1: $L_{\mathrm{m}}(G||S^{\mathrm{F}}) \cap (\Sigma^{\mathrm{N}})^\star \subseteq K^{\mathrm{N}}$

Second, we use the degraded specification $K^{\mathrm{D}} \subseteq \Sigma^\star$ that represents the *admissible* behavior after a fault occurrence. In principle, we want that the system continues its operation after any fault while considering the past system behavior until the fault. That is, a suitable part of the behavior before a fault concatenated with the behavior after a fault should belong to K^{D}. Formally, we want that

P2: it holds for all $s \in L_{\mathrm{m}}(G||S^{\mathrm{F}}) \cap (\Sigma^{\mathrm{N}})^\star \Sigma^{\mathrm{F}} (\Sigma^{\mathrm{N}})^\star$ that there exists a partition $s = s_1^1 s_1^2 \cdots s_k^1 s_k^2 \mathtt{f} s_3$ with $\mathtt{f} \in \Sigma^{\mathrm{F}}$, $s_i^j \in (\Sigma^{\mathrm{N}})^\star$ for $i = 1, \ldots, k$ and $j = 1, 2$, $s_1^1 \cdots s_k^1 \in \overline{K^{\mathrm{N}}}$ and $s_1^2 \cdots s_k^2 s_3 \in K^{\mathrm{D}}$.

In words, $s_1^1 \cdots s_k^1 \in \overline{K^{\mathrm{N}}}$ requires that one part of the substring before a fault occurrence belongs to the nominal behavior, whereas $s_1^2 \cdots s_k^2 s_3 \in K^{\mathrm{D}}$ requires that the remaining substring $s_1^2 \cdots s_k^2$ before the fault occurrence can be continued to a string in K^{D}. That is, a substring of the non-faulty behavior that should originally fulfill K^{N} is used to complete K^{D}. Note that he condition $s_1^1 \cdots s_k^1 \in \overline{K^{\mathrm{N}}}$ is introduced in order to enable system repair as discussed in Section 4.4.

Third, we introduce the faulty behavior specification $K^F \subseteq \Sigma^\star$. This specification represents the *desired* system behavior after fault. That is, we ideally want to achieve K^{F} after any fault occurrence in the sense of converging to K^{F} after a bounded number of event occurrences.

P3: $K^{\mathrm{F}} \Leftarrow L_{\mathrm{m}}(G||S^{\mathrm{F}})/\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}$.

We next combine the previously introduced conditions in order to formulate the fault-recovery problem addressed in this paper.

Problem 2. *Assume that G, Σ_u, Σ, Σ^N, Σ^F, K^N, K^D, K^F are given as above. We want to design a nonblocking fault-recovery supervisor $S^F = (Q^F, \Sigma, v^F, q_0^F, Q_m^F)$ for G and Σ_u such that P1, P2 and P3 hold.*

(P1) $L_m(G||S^F) \cap (\Sigma^N)^\star \subseteq K^N$,

(P2) *it holds for all* $s \in L(G||S^F) \cap (\Sigma^N)^\star \Sigma^F (\Sigma^N)^\star$ *that* $s = s_1^1 s_1^2 \cdots s_k^1 s_k^2 f s_3$ *with* $f \in \Sigma^F$, $s_i^j \in (\Sigma^N)^\star$ *for* $i = 1, \ldots, k$ *and* $j = 1, 2$, $s_1^1 \cdots s_k^1 \in \overline{K^N}$ *and* $s_1^2 \cdots s_k^2 s_3 \in K^D$,

(P3) $K^F \Leftarrow L_m(G||S^F)/\overline{K^N}\Sigma^F$.

We denote the obtained fault-recovery supervisor as nominally optimal *if*

$$L_m(G||S^F) \cap (\Sigma^N)^\star = SupC(K^N, L(G^N), \Sigma_u) \tag{4.1}$$

and as optimally recovering *if*

$$L_m(G||S^F)_\infty = suf(SupC(K^F, L(G), \Sigma_u). \tag{4.2}$$

That is, S^{F} is nominally optimal if it realizes the maximally permissive nominal behavior before any fault occurrence and S^{F} is optimally recovering if it asymptotically realizes the maximally permissive faulty behavior.

4.2 Solution to the Fault-Recovery Problem

In order to solve Problem 2, we first construct a specification language $K^{\mathrm{A}} \subseteq L_{\mathrm{m}}(G)$ that captures conditions (P1) and (P2). To this end, we first note that

$$\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}} \cap L(G)$$

contains all plant strings that follow the nominal specification K^{N} and terminate with a fault and

$$\overline{K^{\mathrm{N}}}\Sigma_{\mathrm{F}}\Sigma^{\star} \cap L(G)$$

contains all plant strings that fulfill the nominal specification until a fault occurrence. Then, applying the interleaving composition in Definition 1, we define the language

$$K^{\mathrm{A}} = (\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}} ||| K^{\mathrm{D}}) \cap (\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star} \cap L_{\mathrm{m}}(G)). \tag{4.3}$$

K^{A} contains all plant strings such that one substring until a fault occurrence belongs to $\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}$, whereas the remaining substring before a fault can be completed to fulfill the degraded specification K^{D}. At the same time, all strings in K^{A} before a fault occurrence belong to $\overline{K^{\mathrm{N}}}$. Next, we compute the supervisor $S^{\mathrm{A}} = (Q^{\mathrm{A}}, \Sigma, \nu^{\mathrm{A}}, q_0^{\mathrm{A}}, Q_{\mathrm{M}}^{\mathrm{A}})$ such that

$$L_{\mathrm{m}}(S^{\mathrm{A}}) = SupC(K^{\mathrm{A}}, L(G), \Sigma_{\mathrm{u}}). \tag{4.4}$$

According to the definition of K^{A}, S^{A} realizes the maximally permissive closed-loop behavior such that $G||S^{\mathrm{A}}$ fulfills the nominal specification before any fault occurrence and continues following the degraded specification after a fault occurs according to (P1) and (P2) in Problem 2.

We finally need to account for (P3). To this end, refer to language convergence introduced in Section 2.6. Using modified language convergence, we propose to com-

pute the supervisor for fault-recovery S^{F} such that

$$L_{\mathrm{m}}(S^{\mathrm{F}}) = \; CA(K^{\mathrm{F}}, L_{\mathrm{m}}(S^{\mathrm{A}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}}) \cup (\overline{CA(K^{\mathrm{F}}, L_{\mathrm{m}}(S^{\mathrm{A}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}})} \cap K^{\mathrm{N}}). \quad (4.5)$$

According to its computation, S^{F} ensures that the closed-loop system $G||S^{\mathrm{F}}$

1. follows the nominal specification K^{N} before any fault occurrence,
2. continues meeting the degraded specification K^{D} after a fault occurs,
3. converges to the faulty behavior specification K^{F}.

We next confirm that the existence of S^{F} according to the described computation in (4.3) to (4.5) is necessary and sufficient for the solution of Problem 2.

Theorem 3. *Consider G, Σ_u, Σ, Σ^N, Σ^F, K^N, K^D, K^F as in Problem 2. Then, a solution to Problem 2 exists if and only if it holds that $L_m(S^F) \neq \emptyset$ for the supervisor S^F according to* (4.5)*. Furthermore, S^F solves Problem 2 if $L_m(S^F) \neq \emptyset$.*

Proof. (IF) We assume that $L_{\mathrm{m}}(S^{\mathrm{F}})$ according to (4.5) is non-empty and we show that S^{F} is a solution to Problem.

(P1) It follows from (4.5) that $CA(K^{\mathrm{F}}, L_{\mathrm{m}}(S^{\mathrm{A}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}}) \neq \emptyset$. Furthermore,

$$\begin{aligned} &CA(K^{\mathrm{F}}, L_{\mathrm{m}}(S^{\mathrm{A}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}}) \cap (\Sigma^{\mathrm{N}})^{\star} \subseteq \\ \subseteq &L_{\mathrm{m}}(S^{\mathrm{A}}) \cap (\Sigma^{\mathrm{N}})^{\star} \subseteq K^{\mathrm{A}} \cap (\Sigma^{\mathrm{N}})^{\star} = \emptyset \end{aligned}$$

Hence, $L_{\mathrm{m}}(S^{\mathrm{F}}) \cap (\Sigma^{\mathrm{N}})^{\star} = \overline{CA(K^{\mathrm{F}}, L_{\mathrm{m}}(S^{\mathrm{A}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}})} \cap K^{\mathrm{N}} \subseteq K^{\mathrm{N}}$ according to (4.5).

(P2) With (4.4), (4.5), it holds that $L_{\mathrm{m}}(S^{\mathrm{F}}) \cap (\Sigma^{\mathrm{N}})^{\star}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star} \subseteq K^{\mathrm{A}}$. That is, $s \in L_{\mathrm{m}}(G||S^{\mathrm{F}}) \cap (\Sigma^{\mathrm{N}})^{\star}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star} \Rightarrow s \in K^{\mathrm{A}} \subseteq (\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}|||K^{\mathrm{D}})$. Then, Definition 1 directly implies that s fulfills (P2).

(P3) Since $\emptyset \neq CA(K^{\mathrm{F}}, L_{\mathrm{m}}(S^{\mathrm{A}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}}) = L_{\mathrm{m}}(S^{\mathrm{F}}) \cap (\Sigma^{\mathrm{N}})^{\star}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star}$, it follows that $K^{\mathrm{F}} \Leftarrow L_{\mathrm{m}}(G||S^{\mathrm{F}})/\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}$.

(ONLY IF) We assume that a solution supervisor $\hat{S}^{\mathrm{F}}$ to Problem 2 exists and we need to show that $L_{\mathrm{m}}(S^{\mathrm{F}})$ according to (4.5) is non-empty.

According to Problem 2, $\hat{S}^{\mathrm{F}}$ fulfills (P1) to (P3). In particular, (P3) implies that $\emptyset \neq L_{\mathrm{m}}(\hat{S}^{\mathrm{F}}) \cap \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star} = CA(K^{\mathrm{F}}, L_{\mathrm{m}}(\hat{S}^{\mathrm{F}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}})$. In addition, we show that $L_{\mathrm{m}}(G||\hat{S}^{\mathrm{F}}) \cap \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star} \subseteq K^{\mathrm{A}}$. Let $s \in L_{\mathrm{m}}(G||\hat{S}^{\mathrm{F}}) \cap \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star}$. Then, by (P2), $s = s_1^1 s_1^2 \cdots s_k^1 s_k^2 \mathtt{f} s_3$ with $\mathtt{f} \in \Sigma^{\mathrm{F}}$, $s_i^j \in (\Sigma^{\mathrm{N}})^{\star}$ for $i = 1, \ldots, k$ and $j = 1, 2$, $s_1^1 \cdots s_k^1 \in \overline{K^{\mathrm{N}}}$ and $s_1^2 \cdots s_k^2 s_3 \in K^{\mathrm{D}}$. Then, Definition 1 implies that $s \in (\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}|||K_D)$. Moreover, by (P1), $s_1^1 s_1^2 \cdots s_k^1 s_k^2 \in \overline{K^{\mathrm{N}}} \subseteq L(G)$ and hence, $s \in \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star} \cap L(G)$. That is, indeed $s \in K^{\mathrm{A}}$.

Considering that $CA(K^{\mathrm{F}}, L_{\mathrm{m}}(\hat{S}^{\mathrm{F}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}})$ is controllable for $L(G)$ and Σ_{u}, $CA(K^{\mathrm{F}}, L_{\mathrm{m}}(\hat{S}^{\mathrm{F}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}}) \subseteq L_{\mathrm{m}}(S^{\mathrm{A}}) = SupC(K^{\mathrm{A}}, L(G), \Sigma_{\mathrm{u}})$. Finally, we conclude that

$$\begin{aligned}\emptyset &\neq CA(K^{\mathrm{F}}, L_{\mathrm{m}}(\hat{S}^{\mathrm{F}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}}) \\ &\subseteq CA(K^{\mathrm{F}}, L_{\mathrm{m}}(S^{\mathrm{A}}), \overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}}) \subseteq L_{\mathrm{m}}(S^{\mathrm{F}}).\end{aligned}$$

□

In order to evaluate the computational complexity of our method, we write y^i for the state count of canonical recognizers for the specifications K^i, $i \in \{\mathrm{N}, \mathrm{D}, \mathrm{F}, \mathrm{A}\}$, $|X|$ for the state count of G and $|Q^{\mathrm{A}}|$ for the state count of S^{A}. Then, the interleaving composition $\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}|||K^{\mathrm{D}}$ in (4.3) requires $O(2^{y^{\mathrm{N}} \cdot y^{\mathrm{D}}})$. Although this complexity is exponential in the respective state counts, it is observed in our example evaluations that the intersection with $(\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star}) \cap L(G)$ mitigates the exponential blow-up. The evaluation of (4.4) is performed with $O(|X|^2 \cdot |Q^{\mathrm{A}}|^2)$ and (4.5) is computed in $O(|Q^{\mathrm{A}}|^2 \cdot 2^{2s^{\mathrm{F}}||})$ according to Section 2.6.

4.3 Application Example

In order to illustrate the concept of fault-recovery as considered in this chapter, we apply the proposed method to the example system which is introduced in Section 2.7. The overview of the system is shown in Figure 11.

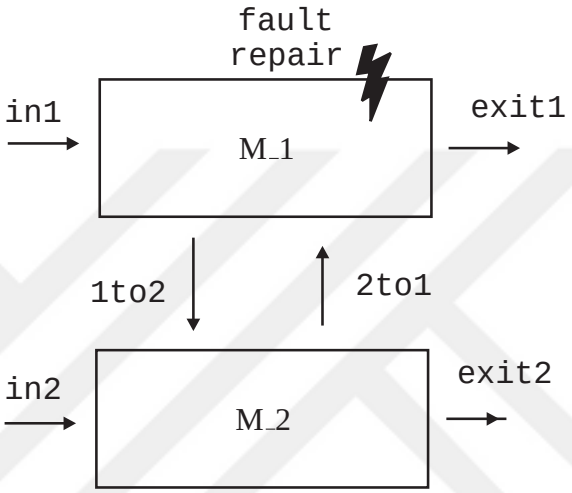


Figure 11: Schematic of the example system with fault events.

In this chapter, it is assumed that a fault (f) that disables the operation of M_1 can occur whenever a product is present. Automata models for the two machines are shown in Fig. 12 such that $G = G_1||G_2$. Furthermore, $\Sigma^{\mathrm{F}} = \{\mathtt{f}\}$ and it is assumed that $\Sigma_{\mathrm{u}} = \{\mathtt{f}\}$ for this example.

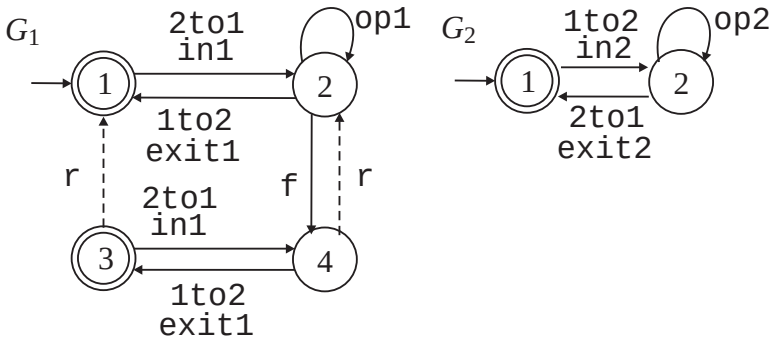


Figure 12: Plant model automata of the example system.

We specify the nominal behavior of the closed-loop system such that only M_1

is used and products that enter M_1 are processed before leaving M_1. The nominal specification is given by $K^{\mathrm{N}} = L_{\mathrm{m}}(C^{\mathrm{N}})$ in Fig. 13. The degraded specification K^{D} is formulated such that products are no longer processed by the faulty machine M_1 but by the other machine M_2. In addition, products in the faulty machine M_1 should remain there. The automata C_1^{D} and C_2^{D} in Fig. 13 address these requirements such that $K^{\mathrm{D}} = L_{\mathrm{m}}(C_1^{\mathrm{D}} || C_2^{\mathrm{D}})$. Finally, we want to achieve that products directly enter M_2 and are processed there in the faulty case. Hence, our faulty behavior specification is $K^{\mathrm{F}} = L_{\mathrm{m}}(C^{\mathrm{F}})$ in Fig. 13.

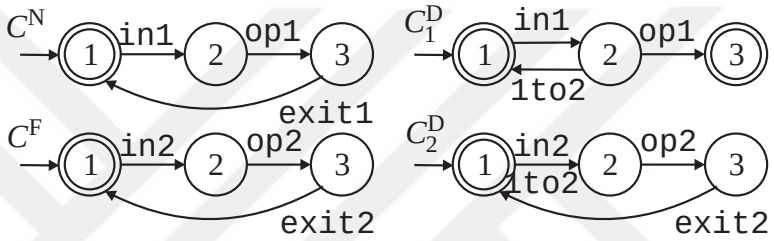


Figure 13: Nominal specification automaton C^{N} (alphabet Σ); degraded specification automata C_1^{D} (alphabet $\Sigma \setminus \{\mathtt{in2}, \mathtt{op2}, \mathtt{exit2}\}$), C_2^{D} (alphabet $\Sigma \setminus \{\mathtt{in1}, \mathtt{op1}\}$) and faulty specification automaton C^{F} (alphabet Σ).

Using G (without the r-transitions), Σ_{u}, K^{N}, K^{D}, K^{F}, Σ^{F} as introduced in this section, it is possible to evaluate the fault-recovery supervisor S^{F} based on (4.3), (4.4) and (4.5). The result is shown in Fig. 14. Hereby, the left-hand part of S^{F} represents the nominal behavior of the example system, whereby it can be verified that $L_{\mathrm{m}}(S^{\mathrm{F}}) \cap (\Sigma^{\mathrm{N}})^{\star} \subseteq K^{\mathrm{N}}$. Moreover, it turns out for this simple example that convergence to the faulty behavior specification is achieved immediately (right-hand part of S^{F}).

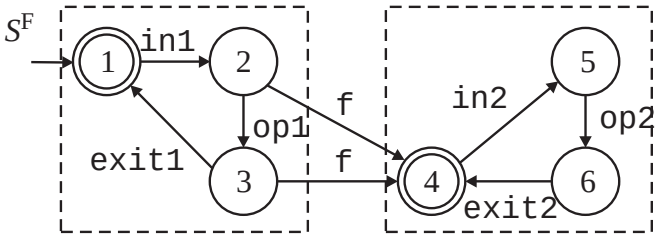


Figure 14: Fault-recovery supervisor S^{F}.

4.4 Handling System Repair

We next show that the same computation as in Section 4.2 can be used in order to handle system repair after a fault. To this end, we define a set of repair events Σ^{R} and consider a plant model G that allows for repair after a fault as is shown in Fig. 12. In addition, we introduce a repair specification K^{R} that continues the system behavior after a repair happened. Using this model, we intend to achieve the following behavior that is complementary to the behavior specified in Problem 2:

R1: Follow the fault-recovering behavior that is represented by $L_{\mathrm{m}}(G||S^{\mathrm{F}})$ as computed in Section 4.2 until a repair is performed,

R2: Continue the system behavior according to the repair specification K^{R} after repair,

R3: Finally converge to the nominal specification K^{N}.

It is readily observed that (R1) to (R3) above are obtained by substitution from item (P1) to (P3) in the formulation of Problem 2 as follows. $\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}$ is replaced by $\overline{K^{D}}\Sigma^{\mathrm{R}}$, K^{D} is replaced by K^{R}, K^{F} is replaced by K^{N} and $\overline{K^{\mathrm{N}}}\Sigma^{\mathrm{F}}(\Sigma^{\mathrm{N}})^{\star}$ is replaced by $L(G||S^{\mathrm{F}})\Sigma^{\mathrm{R}}(\Sigma^{\mathrm{N}})^{\star}$. That is, the same solution procedure can be applied, first computing a specification

$$K^{\mathrm{B}} = [\overline{K^{D}}\Sigma^{\mathrm{R}}|||K^{\mathrm{R}}) \cap (L(G||S^{\mathrm{F}})\Sigma^{\mathrm{R}}(\Sigma^{\mathrm{N}})^{\star} \cap L_{\mathrm{m}}(G))]. \tag{4.6}$$

Next, the supervisor S^{B} is computed with

$$L_{\mathrm{m}}(S^{\mathrm{B}}) = SupC(K^{\mathrm{B}}, L(G), \Sigma_{\mathrm{u}} \setminus \Sigma^{\mathrm{F}}). \tag{4.7}$$

Finally, convergence to the nominal behavior is achieved by computing a supervisor for repair S^{R} such that

$$\begin{array}{ll} L_{\mathrm{m}}(S^{\mathrm{R}}) & = CA(K^{\mathrm{N}}, L_{\mathrm{m}}(S^{\mathrm{B}}), L(G||S^{\mathrm{F}})\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}} \setminus \Sigma^{\mathrm{F}}) \cup \\ L_{\mathrm{m}}(G||S^{\mathrm{F}}) \cap & \overline{CA(K^{\mathrm{N}}, L_{\mathrm{m}}(S^{\mathrm{B}}), L(G||S^{\mathrm{F}})\Sigma^{\mathrm{F}}, \Sigma_{\mathrm{u}} \setminus \Sigma^{\mathrm{F}})} \end{array} \tag{4.8}$$

Note that the uncontrollable events $\Sigma_{\mathrm{u}} \setminus \Sigma^{\mathrm{F}}$ are used in (4.7) and (4.8). Here, we consider the events in Σ^{F} as controllable since we compute the supervisor S^{R} for the case after repair but without additional fault. The possibility of multiple fault occurrences is discussed below.

In our example, the model G in Fig. 12 already includes a system repair (dashed events `r`). Furthermore, we would like to achieve the specification K^{B} that is given in Fig. 15 after repair is performed. That is, products that are processed by M_2 should move to M_1 and any new products should enter from M_1 and should be processed there.

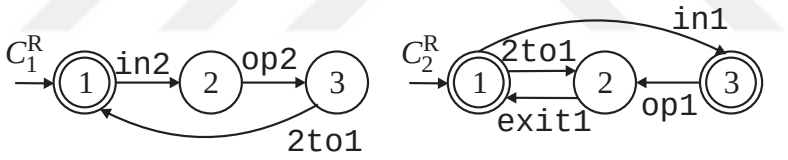


Figure 15: Specification automata C_1^{R} and C_2^{R} after repair.

Using the computation described in this section, we arrive at the supervisor for repair S^{R} as shown in Fig. 16. It can be seen that the upper left-hand part of S^{R} constitutes the nominal behavior, whereas the upper right-hand part represents the behavior after a fault. The lower part of S^{R} is is the desired behavior after repair. That is, products are delivered to and operated at M_1 and finally the nominal behavior is recovered.

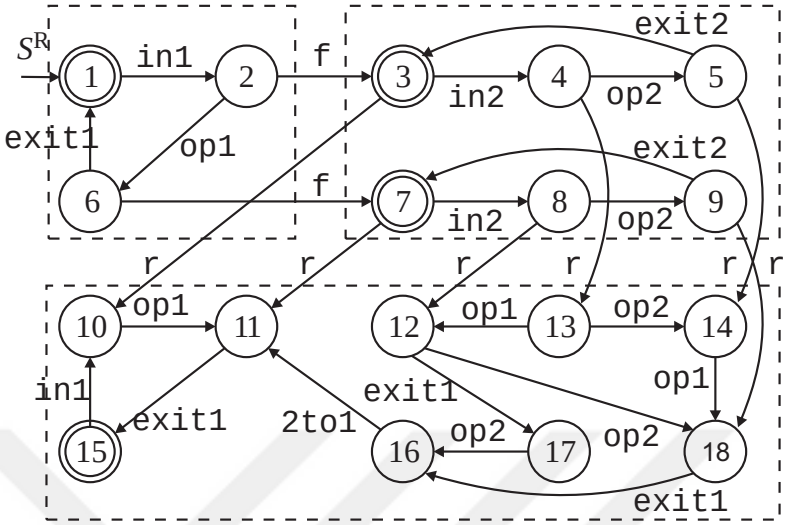


Figure 16: Supervisor after repair S^R.

4.5 Arbitrary Fault Occurrences and System Repairs

The computation of S^R as in (4.8) assumes that no further fault can happen after repair, since no fault is allowed after repair according to (4.6). In practice, this is not a realistic assumption. In order to determine a fault-recovery supervisor that allows arbitrary faults and repairs, we propose to iteratively apply the computation for fault-recovery and repair until no new behavior is added to the resulting supervisor. To this end, we first write a generalized representation of the computation in (4.3) to (4.5) ((4.6) to (4.8)) depending on the input languages $K_1,\ldots,K_4$ and alphabets Σ_1,Σ_2.

$$K^C = [(\overline{K_1}\Sigma_1 ||| K_2) \cap (\overline{K_3}\Sigma_1(\Sigma^N)^\star \cap L(G))], \tag{4.9}$$

$$L_m(\hat{S}^C) = SupC(K^C, L(G), \Sigma_u \setminus \Sigma_2), \tag{4.10}$$

$$L_m(S^C) = CA(K_4, L_m(\hat{S}^C), \overline{K_3}\Sigma_2, \Sigma_u \setminus \Sigma_2) \cup K_3 \cap \overline{CA(K_4, L_m(\hat{S}^C), \overline{K_3}\Sigma_2, \Sigma_u \setminus \Sigma_2)} \tag{4.11}$$

We define the supervisors $S_i^F = (Q_i^F, \Sigma, v_i^F, q_{0,i}^F, Q_{m,i}^F)$ and $S_i^R = (Q_i^R, \Sigma, v_i^R, q_{0,i}^R, Q_{m,i}^R)$ in iteration i and write $S_{i,q}^F$ ($S_{i,q}^R$) for the supervisor S_i^F (S_i^R) starting from state $q \in Q_i^F$ ($q \in Q_i^R$). Then, we initialize the iterative process with $i := 1$, $S_1^F := S^F$ in (4.5) and

$S_1^{\mathrm{R}} := S^{\mathrm{R}}$ in (4.8) and we apply the following steps.

1. $i := i+1$,
2. Compute S_i^{F} using (5.1) to (5.3) with the inputs $K_1 = K^{\mathrm{R}}$, $K_2 = K^{\mathrm{D}}$, $K_3 = L_{\mathrm{m}}(G||S_{i-1}^{\mathrm{R}})$, $K_4 = K^{\mathrm{F}}$, $\Sigma_1 = \Sigma^{\mathrm{F}}$ and $\Sigma_2 = \Sigma^{\mathrm{R}}$,
3. Define $W_k^{\mathrm{F}} := \{q \in Q_i^{\mathrm{F}} | \exists u \in (\Sigma \setminus \Sigma^{\mathrm{F}})^\star [\Sigma^{\mathrm{F}}(\Sigma^\star \setminus \Sigma^{\mathrm{F}})]^k$ such that $q = \nu_i^{\mathrm{F}}(q_{0,i}^{\mathrm{F}}, u)\}$,
4. Terminate if for all $q \in W_i^{\mathrm{F}}$, there exists a $\hat{q} \in W_{i-1}^{\mathrm{F}}$ such that $L_{\mathrm{m}}(S_{i,q}^{\mathrm{F}}) = L_{\mathrm{m}}(S_{i,\hat{q}}^{\mathrm{F}}) \cap (\Sigma^{\mathrm{N}})^\star$. In that case, transitions leading to q in S_i^{F} are lead to the state $\hat{q}$ instead. Denote the resulting automaton as S,
5. Compute S_i^{R} using (5.1) to (5.3) with the inputs $K_1 = K^{\mathrm{D}}$, $K_2 = K^{\mathrm{R}}$, $K_3 = L_{\mathrm{m}}(G||S_i^{\mathrm{F}})$, $K_4 = K^{\mathrm{N}}$, $\Sigma_1 = \Sigma^{\mathrm{R}}$ and $\Sigma_2 = \Sigma^{\mathrm{F}}$,
6. Define $W_k^{\mathrm{R}} := \{q \in Q_i^{\mathrm{R}} | \exists u \in (\Sigma \setminus \Sigma^{\mathrm{R}})^\star [\Sigma^{\mathrm{R}}(\Sigma \setminus \Sigma^{\mathrm{R}})^\star]^{k-1}$ such that $q = \nu_i^{\mathrm{R}}(q_{0,i}^{\mathrm{R}}, u)\}$,
7. Terminate if for all $q \in W_i^{\mathrm{R}}$, there exists a $\hat{q} \in W_{i-1}^{\mathrm{R}}$ such that $L_{\mathrm{m}}(S_{i,q}^{\mathrm{R}}) = L_{\mathrm{m}}(S_{i,\hat{q}}^{\mathrm{R}}) \cap (\Sigma^{\mathrm{N}})^\star$. In that case, transitions leading to q in S_i^{R} are lead to the state $\hat{q}$ instead. Denote the resulting automaton as S,
8. go to 1.

The algorithm repeatedly computes supervisors for fault-recovery and repair analogous to Section 4.2 and 4.4, whereby the algorithm terminates if the newly added behavior after fault (repair) is identical to the one added in the previous iteration. To this end, it is checked in steps (4) and (7) if the added behavior obtained in step i is identical to the added behavior in step $i-1$. In the positive case, it is possible to lead all incoming transitions of newly added states back to the corresponding state that was added in the previous iteration. The resulting supervisor after termination of the described algorithm handles arbitrary numbers of faults and repairs. In addition, the conditions in Problem 2 are fulfilled for each fault occurrence, whereas the analogous conditions for system repair are fulfilled for each system repair. Although termination of the algorithm was observed for all our examples, a formal result regarding the

termination of our algorithm is subject of future work. As a note, the algorithm is implemented in [23].

For our example, the algorithm terminates in the second iteration with the supervisor S in Fig. 17. It can be seen by inspection that the desired specifications after fault and repair are fulfilled.

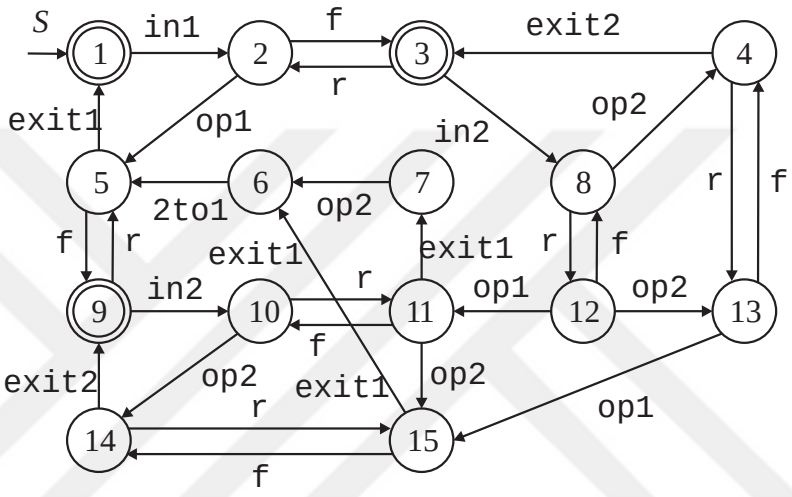


Figure 17: Supervisor S that solves the fault-recovery problem with system repair.

CHAPTER 5

DISCRETE EVENT SUPERVISOR DESIGN AND APPLICATION FOR MANUFACTURING SYSTEMS WITH ARBITRARY FAULTS AND REPAIRS

This chapter extends the method for the fault-recovery and repair of single faults as described in Chapter 4 to the case of different faults. As a result, we obtain a supervisor that follows the specified nominal system behavior in the fault-free case, converges to a desired degraded behavior for each fault type and recovers the nominal behavior after repair. The chapter is organized as follows. Section 5.1 gives a motivating example and Section 5.2 presents the extended problem formulation. Section 5.3 proposes an algorithm for the supervisor computation. We note that the results presented in this chapter are published in the conference paper [7].

5.1 Motivating Example

We recall the same example which is introduced in Section 2.7 in order to illustrate the problem setting in this chapter.

We assume that two faults can happen in the system the main difference from the study in Chapter 4. `F1` can occur in M_1 whenever a product is present and disables the operation of M_1. Likewise, `F2` can occur whenever a product is present in M_2 and disables the operation of M_2. The corresponding repair events are `R1` and `R2`. `R1` can occur after `F1` and `R2` can occur after `F2`. When repair events happen, the correct operation of the respective machine is restored. Automata models G_1 and G_2 for the two machines are shown in Fig. 19. In this chapter, we assume that the two faults do not happen simultaneously which is captured by the automaton F in Fig. 19. That is, the overall plant model is $G = G_1||G_2||F$. Moreover, $\Sigma^F = \{\texttt{F1}, \texttt{F2}\}$,

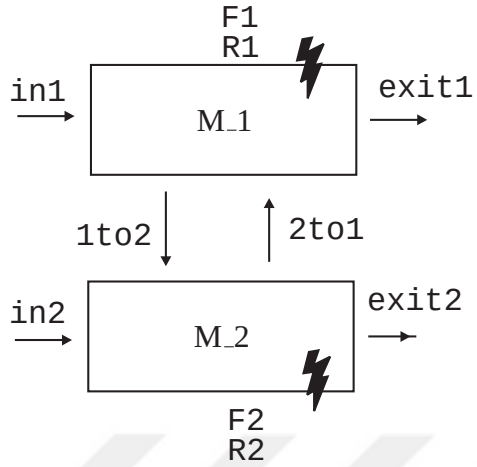


Figure 18: Schematic of the example system

$\Sigma^{\mathrm{R}} = \{\mathtt{R1}, \mathtt{R2}\}$ and it is assumed that $\Sigma_{\mathrm{u}} = \{\mathtt{F1}\ , \mathtt{F2}\}$.

We specify the nominal behavior of the closed-loop system such that products enter one of the machines M_1 or M_2 and are processed by the respective machine before leaving. The nominal specification is given by $K^{\mathrm{N}} = L_{\mathrm{m}}(C^{\mathrm{N}})$ in Fig. 19. In case of a fault in one of the machines, it is desired that the other machine continues the operation. Different from our previous work, this implies that a different behavior should be achieved after different faults.

Considering F1, it is required that products are no longer processed by the faulty machine M_1 but by the other machine M_2. That is, after F1, unprocessed products in M_1 should be moved to M_2 as is modeled by the degraded specification $K^{\mathtt{D1}} = L_{\mathrm{m}}(C_1^{\mathtt{D1}} || C_2^{\mathtt{D1}})$ in Fig. 19. Finally, products should only be handled by M_2 as described by $K^{\mathtt{F1}}$. In case of repair, the operation of M_1 should resume and the system should return to the nominal operation. This behavior is represented by the repair specification $K^{\mathtt{R1}} = L_{\mathrm{m}}(C_1^{\mathtt{R1}} || C_2^{\mathtt{R1}})$ in Fig. 19.

Considering F2, it is required that products are no longer processed by the faulty machine M_2 but by the other machine M_1. The respective specifications $K^{\mathtt{D2}}$ (degraded), $K^{\mathtt{F2}}$ (faulty behavior) and $K^{\mathtt{R2}}$ (repair) are readily obtained from $K^{\mathtt{D1}}$, $K^{\mathtt{F1}}$, $K^{\mathtt{R1}}$ due to the symmetry of the problem (Fig. 19).

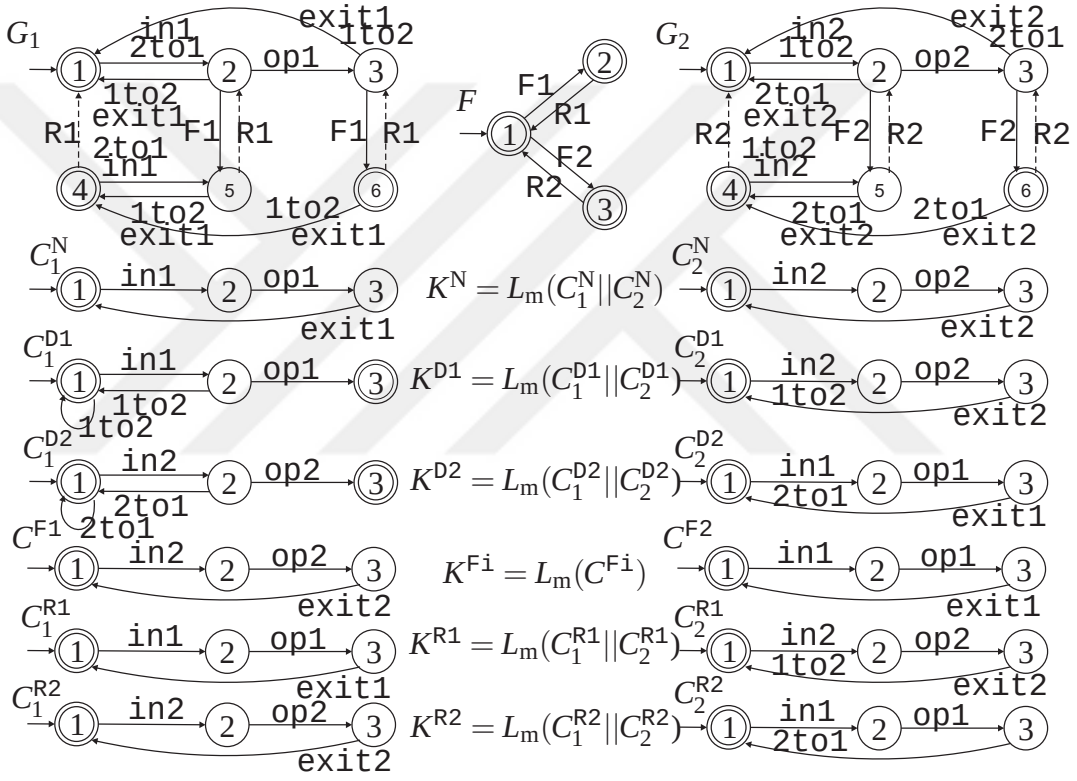


Figure 19: Automata models for the example system: Plant $G = G_1||G_2||F$; nominal specification $K^{\mathrm{N}} = L_{\mathrm{m}}(C_1^{\mathrm{N}}||C_2^{\mathrm{N}})$.

We note that the described setting resembles the setting in Chapter 4. Nevertheless, we now consider that a different behavior is required after different faults which is not captured in Chapter 4.

5.2 Problem Formulation

We next formalize the extended problem setting. The system behavior is characterized by the plant model $G = (X, \Sigma, \delta, x_0, X_\mathrm{m})$, assuming that different faults cannot occur simultaneously. The alphabets $\Sigma_\mathrm{u}, \Sigma^\mathrm{N} \subseteq \Sigma$ are used as before. In addition, we write $\Sigma^\mathrm{F} = \{\mathtt{F1}, \ldots, \mathtt{Fn}\}$ for the fault events and $\Sigma^\mathrm{R} = \{\mathtt{R1}, \ldots, \mathtt{Rn}\}$ for the repair events.

In order to specify the desired system behavior for our fault-recovery and repair problem, we consider the nominal specification K^N as in Section 4.1 and three specifications for each fault $\mathtt{Fi} \in \Sigma^\mathrm{F}$: $K^{\mathtt{Di}}$ (degraded specification); $K^{\mathtt{Fi}}$ (faulty behavior specification); $K^{\mathtt{Ri}}$ (repair specification). These specifications have the same meaning as in Section 4.1 and 4.4 but are now fault-specific.

Using the previously introduced notation, we want to solve the following problem.

Problem 3. *Assume that G, Σ_u, Σ^N, Σ^F, Σ^R, K^N, $K^{\mathtt{Di}}$, $K^{\mathtt{Fi}}$, $K^{\mathtt{Ri}}$ for $\mathtt{Fi} \in \Sigma^F$ are given as above. We want to design a nonblocking fault-recovery and repair supervisor $S = (Q, \Sigma, v, q_0, Q_m)$ for G and Σ_u such that the below conditions (1) to (3) hold. Hereby, we define $L_l = (\Sigma^N)^\star \left(\Sigma^F (\Sigma^N)^\star \Sigma^R (\Sigma^N)^\star\right)^l \cap L(G||S)$ as the set of all closed-loop strings with l faults and repairs.*

(1) $L_m(G||S) \cap (\Sigma^N)^\star \subseteq K^N$.

(2) it holds for all $\mathtt{Fi} \in \Sigma^F$ *and* $l \in \mathbb{N}_0$*:*

a) $\forall s \in L_m(G||S) \cap L_l \mathtt{Fi} (\Sigma^N)^\star$*, there exists a partition* $s = s_1^1 s_1^2 \cdots s_k^1 s_k^2 \mathtt{Fi} s_3$, $s_i^j \in \Sigma^\star$ *for* $i = 1, \ldots, k$ *and* $j = 1, 2$, $s_1^1 \cdots s_k^1 \in L_l$ *and* $s_1^2 \cdots s_k^2 s_3 \in K^{\mathtt{Di}}$.

b) $K^{\mathtt{Fi}} \Leftarrow [L_m(G||S) / (L(G||S) \cap L_l \mathtt{Fi})] \cap (\Sigma^N)^\star$

(3) it holds for all $\texttt{Ri} \in \Sigma^R$ *and* $l \in \mathbb{N}_0$*:*

a) $\forall s \in L_m(G||S^F) \cap L_l\texttt{Fi}(\Sigma^N)^\star\texttt{Ri}(\Sigma^N)^\star$*, there exists a partition* $s = s_1^1 s_1^2 \cdots s_k^1 s_k^2 \texttt{Fi} s_3$*,* $s_i^j \in \Sigma^\star$ *for* $i = 1,\ldots,k$ *and* $j = 1,2$*,* $s_1^1 \cdots s_k^1 \in L(G||S) \cap L_l\texttt{Fi}(\Sigma^N)^\star$ *and* $s_1^2 \cdots s_k^2 s_3 \in K^{\texttt{Ri}}$*.*

b) $K^N \Leftarrow [L_m(G||S)/(L(G||S) \cap L_l\texttt{Fi}(\Sigma^N)^\star\texttt{Ri})] \cap (\Sigma^N)^\star$

The three conditions in Problem 2 can be explained as follows: (1) requires that the closed-loop system fulfills the nominal specification K^{N} in case no fault occurred. (2) states that the closed-loop system should a) continue its operation while fulfilling the corresponding degraded specification $K^{\texttt{Di}}$ and b) converge to the corresponding faulty behavior specification $K^{\texttt{Fi}}$ after each fault occurrence $\texttt{Fi}$. Analogously, (3) states that the closed-loop system should a) continue its operation while fulfilling the corresponding repair specification $K^{\texttt{Ri}}$ and b) converge to the nominal specification $K^{\texttt{N}}$ after each repair $\texttt{Ri}$.

5.3 Supervisor Computation for Different Faults

This section presents our algorithm for the computation of fault-recovery and repair supervisors that handle arbitrary occurrences of different faults as the main contribution of this thesis. To this end, we first provide a generalized representation of the computation in (4.3) to (4.5) and ((4.6) to (4.8)) depending on the input languages $K_1,\ldots,K_4$ and alphabets Σ_1,Σ_2:

$$K^{\mathrm{C}} = [(\overline{K_1}\Sigma_1 ||| K_2) \cap (\overline{K_3}\Sigma_1(\Sigma^{\mathrm{N}})^\star \cap L(G))], \tag{5.1}$$

$$L_{\mathrm{m}}(\hat{S}^{\mathrm{C}}) = SupC(K^{\mathrm{C}}, L(G), \Sigma_{\mathrm{u}} \setminus \Sigma_2), \tag{5.2}$$

$$L_{\mathrm{m}}(S^{\mathrm{C}}) = CA(K_4, L_{\mathrm{m}}(\hat{S}^{\mathrm{C}}), \overline{K_3}\Sigma_2, \Sigma_{\mathrm{u}} \setminus \Sigma_2) \cup K_3 \cap \overline{CA(K_4, L_{\mathrm{m}}(\hat{S}^{\mathrm{C}}), \overline{K_3}\Sigma_2, \Sigma_{\mathrm{u}} \setminus \Sigma_2)} \tag{5.3}$$

Using this representation, we propose to iteratively compute the supervisor $S = (Q, \Sigma, \nu, q_0, Q_{\mathrm{m}})$ in Problem 3. In this computation, we use intermediate supervisors

denoted as $S(r)$, whereby $r \in (\Sigma^{\mathrm{F}}\Sigma^{\mathrm{R}})^{\star}(\varepsilon + \Sigma^{\mathrm{F}})$. For example $S(r)$ for $r = \mathtt{F1R1F2}$ denotes a supervisor that is computed for the successive occurrence of fault $\mathtt{F1}$, repair $\mathtt{R1}$ and fault $\mathtt{F2}$. Furthermore, writing $r = \sigma_1\sigma_2\cdots\sigma_k$, we define $W(r) = \{q \in Q | \exists u \in (\Sigma^{\mathrm{N}})^{\star}\sigma_1(\Sigma^{\mathrm{N}})^{\star}\sigma_2(\Sigma^{\mathrm{N}})^{\star}\cdots(\Sigma^{\mathrm{N}})^{\star}\sigma_k$ such that $q = \nu(q_0, u)\}$ as the set of states in S that are reachable after a sequence of faults and repairs given by r.

Based on this notation, we propose Algorithm 2.

This algorithm repeatedly computes supervisors for fault-recovery and repair following the tree in Fig. 20. We note that the tree is shown for the case of two faults for ease of representation. We believe that the general representation for an arbitrary number of faults is straightforward. This tree captures all combinations of successive fault occurrences and repairs, respecting the assumption that different faults do not occur simultaneously. Hereby, the set $\mathscr{A}$ keeps track of the supervisors that belong to the leaves of the tree. This set is initialized with the supervisors for one fault and one repair in line 1 to 3 of Algorithm 2. The algorithm then successively takes supervisors from $\mathscr{A}$ (line 7) and computes new supervisors according to (5.1) to (5.3). For example, the supervisor $S(\mathtt{F1\,R1\,F2})$ is computed from $S(\mathtt{F1\,R1})$ using (5.1) to (5.3) with the specifications $K_1 = K^{\mathtt{R1}}$, $K_2 = K^{\mathtt{D2}}$, $K_3 = L_{\mathrm{m}}(G||S(\mathtt{F1\,R1}))$, $K_4 = K^{\mathtt{F2}}$ and the alphabets $\Sigma_1 = \{\mathtt{F2}\}$ and $\Sigma_2 = \{\mathtt{R1}\}$. This computation is performed in line 8 in Algorithm 2. The overall supervisor is then updated by adding the newly found behavior (line 9). In the next step, it is checked if really new behavior is found in the current iteration (line 10 to 15). For example, again consider $S(\mathtt{F1\,R1\,F2})$. Then, $W(\mathtt{F1\,R1\,F2})$ represents the states in S after the occurrence of $\mathtt{F2}$ and $L_{\mathrm{m}}(S_q)$ for each $q \in W(\mathtt{F1\,R1\,F2})$ represents the possible behavior after $\mathtt{F2}$ from state q. If this behavior is also obtained after $\mathtt{F2}$ in the supervisor $S(\mathtt{F2})$, this implies that $S(\mathtt{F1\,R1\,F2})$ does not realize any new behavior after $\mathtt{F2}$. Hence, it is possible to discard $S(\mathtt{F1\,R1\,F2})$ and continue the behavior after $\mathtt{F2}$ with the previously computed $S(\mathtt{F2})$ instead and $S(\mathtt{F1\,R1\,F2})$ is not inserted in $\mathscr{A}$ (line 12). If new behavior is found, $S(\mathtt{F1\,R1\,F2})$ is inserted as a new leaf in $\mathscr{A}$. After that, the same procedure is performed for the case of a repair in line 16 to 23. The algorithm terminates if no more leaves are inserted in $\mathscr{A}$, which happens as soon as no new behavior is found.

input : G, Σ_u, K^N, Σ^F, $K^{\mathtt{Di}}$, $K^{\mathtt{Fi}}$, $\mathtt{Fi} \in \Sigma^F$, Σ^R, $\Sigma^{\mathtt{Ri}}$, $\mathtt{Ri} \in \Sigma^R$
output: S

Compute $S(\mathtt{Fi})$ for each $\mathtt{Fi} \in \Sigma^F$ (as in Section **??**)
Compute $S(\mathtt{Fi}\,\mathtt{Ri})$ for each $\mathtt{Ri} \in \Sigma^R$ (as in Section 4.4)
$\mathscr{A} = \bigcup_{\mathtt{Fi} \in \Sigma^F} \{S(\mathtt{Fi}), S(\mathtt{Fi}\,\mathtt{Ri})\}$
Compute S such that $L_m(S) = \bigcup_{\hat{S} \in \mathscr{A}} L_m(\hat{S})$
while $\mathscr{A} \neq \emptyset$ **do**
  for $\mathtt{Fi} \in \Sigma^F$ *and* $S(r\mathtt{Fj}\mathtt{Rj}) \in \mathscr{A}$ **do**
    Remove $S(r\mathtt{Fj}\mathtt{Rj})$ from $\mathscr{A}$
    Compute $S(r\mathtt{Fj}\mathtt{Rj}\mathtt{Fi})$ using (5.1) to (5.3) with $K_1 = K^{\mathtt{Rj}}$, $K_2 = K^{\mathtt{Di}}$, $K_3 = L_m(G||S(r\mathtt{Rj}))$, $K_4 = K^{\mathtt{Fi}}$, $\Sigma_1 = \{\mathtt{Fi}\}$ and $\Sigma_2 = \{\mathtt{Rj}\}$
    Compute new S such that $L_m(S) = L_m(S) \cup L_m(S(r\mathtt{Fj}\mathtt{Rj}\mathtt{Fi}))$
    Compute $W(r\mathtt{Fi})$ and $W(r\mathtt{Fj}\mathtt{Rj}\mathtt{Fi})$
    if $\forall q \in W(r\mathtt{Fj}\mathtt{Rj}\mathtt{Fi})$, $\exists \hat{q} \in W(r\mathtt{Fi})$ *such that* $L_m(S_q) = L_m(S_{\hat{q}}) \cap (\Sigma^N)^\star$ **then**
      Connect transitions leading to q in S to the state $\hat{q}$ instead
    else
      $\mathscr{A} = \mathscr{A} \cup \{S(r\mathtt{Fj}\mathtt{Rj}\mathtt{Fi})\}$
    end
    Compute $S(\mathtt{Fj}\mathtt{Rj}\mathtt{Fi}\mathtt{Ri})$ using (5.1) to (5.3) with the inputs $K_1 = K^{\mathtt{Di}}$, $K_2 = K^{\mathtt{Ri}}$, $K_3 = L_m(G||S(\mathtt{Fj}\mathtt{Rj}\mathtt{Fi})$, $K_4 = K^N$, $\Sigma_1 = \{\mathtt{Ri}\}$ and $\Sigma_2 = \{\mathtt{Fi}\}$
    Compute new S such that $L_m(S) = L_m(S) \cup L_m(S(r\mathtt{Fj}\mathtt{Rj}\mathtt{Fi}\mathtt{Ri})$
    Compute $W(r\mathtt{Fi}\mathtt{Ri})$ and $W(r\mathtt{Fj}\mathtt{Rj}\mathtt{Fi}\mathtt{Ri})$.
    if $\forall q \in W(r\mathtt{Fj}\mathtt{Rj}\mathtt{Fi}\mathtt{Ri})$, $\exists \hat{q} \in W(r\mathtt{Fi}\mathtt{Ri})$ *such that* $L_m(S_q) = L_m(S_{\hat{q}}) \cap (\Sigma^N)^\star$ **then**
      Connect transitions leading to q in S to the state $\hat{q}$ instead
    else
      $\mathscr{A} = \mathscr{A} \cup \{S(r\mathtt{Fj}\mathtt{Rj}\mathtt{Fi}\mathtt{Ri})\}$
    end
  end
end

Algorithm 2: Computation of S for different repeated faults.

The resulting supervisor after termination of Algorithm 2 handles arbitrary numbers and sequences of faults and repairs.

We next briefly outline that the conditions of Problem 3 are fulfilled by S computed in Algorithm 2. (1) follows from the initialization in line 1 and 2. (2) is addressed in line 8 and 9 and is preserved by the re-connection of transitions in line 12. Similarly, (3) is addressed in line 16, 17 and 20. Finally, it has to be noted that we do not have a formal result regarding the termination of the algorithm at the current stage of our research. Nevertheless, we note that termination was achieved in all our example evaluations.

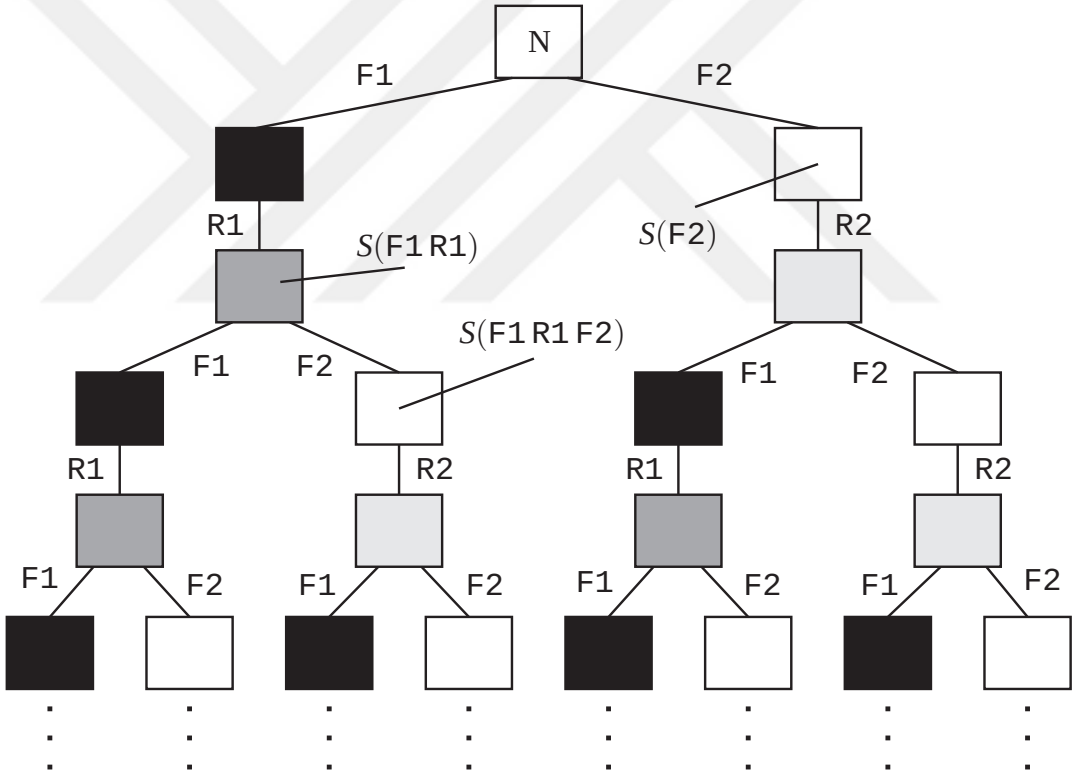


Figure 20: Tree for the computation of S.

We applied Algorithm 2 to the example with two faults in Section 5.1. The resulting tree is shown in Fig. 21. That is, no new behavior is obtained when computing $S(\texttt{F1 R1 F1})$ (compared to $S(\texttt{F1})$, $S(\texttt{F1 R1 F2})$ (compared to $S(\texttt{F2})$, $S(\texttt{F2 R2 F1})$ (compared to $S(\texttt{F1})$ and $S(\texttt{F2 R2 F2})$ (compared to $S(\texttt{F2})$. The final supervisor S for

our example has 79 states and is too large to be shown in this thesis.

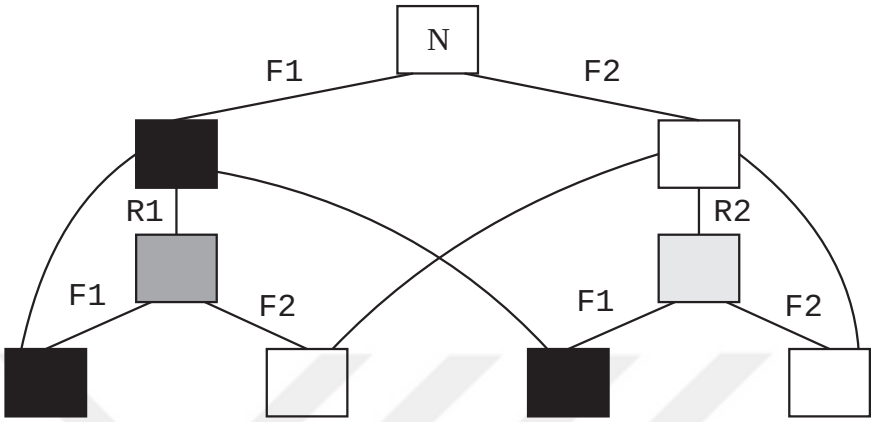


Figure 21: Tree obtained for the example system. Termination is achieved after F1 R1 F1, F1 R1 F2, F2 R2 F1 and F2 R2 F2.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

The occurrence of a fault indicates the deviation of a dynamic system's behavior from its desired (nominal) behavior and usually has a negative impact. In this context, fault-tolerant and fault-recovery control are concerned with designing controllers that enable the operation of a system even in case of faults. The computation of fault-tolerant supervisors for discrete event systems (DES) and the fault-recovery and repair of discrete event systems (DES) are the main subjects of the thesis.

Regarding fault-tolerance, we address faults modeled by faulty events, the occurrence of which is no longer possible in case of a fault and we want the closed-loop system to fulfill a given specification even in case of fault. In this modeling framework, an algorithm for the verification of fault tolerance of a given specification is our first contribution. Second, we prove the existence of a supremal fault-tolerant sublanguage, in case a given specification is not fault-tolerant. As the third contribution, we suggest a polynomial-time algorithm for the computation of the supremal fault-tolerant sublanguage.

Regarding fault-recovery and repair, we first propose a new method for the design of fault-recovery supervisors based on the concepts such as interleaving composition and language convergence. Our fault-recovery supervisor follows the nominal system behavior as long as no fault occurs, and switches to a degraded behavior once a fault occurs. Afterwards, the closed-loop behavior converges to a desired specification under fault in a bounded number of transitions. Unlike the existing literature, the computational method for our fault-recovery supervisor can also be used to con-

duct system repair. Finally, an iterative application of our method enables computing supervisors for the repetitive occurrence of faults and system repairs.

In addition, this thesis addresses the case where, after each fault, different faults can occur and a different behavior is required after each fault. To this end, we propose a new algorithm that iteratively computes a fault-recovery and repair supervisor following a tree of the possible sequences of fault and repair. After termination, the computed supervisor allows arbitrary sequences of faults and repairs. All methods and algorithms are illustrated by small manufacturing system examples.

6.2 Future Work

The work presented in this thesis assumes full event observation, that is, all events can directly be observed/measured. In faulty systems, it can be expected that some events cannot be directly observed. Hence, an interesting subject for future work is the extension of the proposed methods to the case of DES under partial observation.

REFERENCES

1. **Cassandras C.G., Lafortune S. (2008)**, "*Introduction to discrete event systems, Second Edition",* Springer .

2. **Blanke M., Kinnaert M., Lunze J., Staroswiecki., (2010),** *"Diagnosis and Fault-Tolerant Control",* Springer.

3. **Sülek A.N., Schmidt K.W., (2013),** "*Computation of fault-tolerant supervisors for discrete event systems",* 4th IFAC Workshop on Dependable Control of Discrete Systems, York, p.115–120.

4. **Hoare C.A.R., (1995),** "*Communicating Sequential Processes",* Prentice Hall International

5. **Willner Y.M., Heymann M. (1995)**, "*Language convergence in controlled discreteevent systems",* IEEE Transactions on Automatic Control, vol.40, no.4, p. 616–627.

6. **Sülek A.N., Schmidt K.W., (2014),** "*Computation of supervisors for fault-recovery and repair for discrete event systems",* Discrete Event Systems Workshop, Cachen, p.428–433.

7. **Sülek A.N., Schmidt K.W., (2015),** *"Discrete event supervisor design and application for manufacturing systems with arbitrary faults and repairs",* IEEE International Conference on Automation Science and Engineering, Gothenburg.

8. **Saboori A., Hashtrudi-Zad S., (2005)**, "Fault recovery in discrete event systems", In Proc. Computational Intelligence: Methods and Applications.

9. **Paoli A., Lafortune S., (2005),** *"Safe diagnosability for fault tolerant supervision of discrete event systems",* Automatica, vol.41, no.8, p.1335–1347.

10. **Paoli A., Sartini M., Lafortune S.,(2011),** ***"****Active fault tolerant control of discrete event systems using online diagnostics",* Automatica, vol.47, no.4, p.639–649.

11. **Kumar R., Takai S., (2012),** *"A framework for control-reconfiguration following fault-detection in discrete event systems",* 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, Mexico City, p.848–853.

12. **Wittmann T., Richter J., Moor T., (2012),** *"Fault-tolerant control of discrete event systems based on fault-accommodating models",* 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, Mexico City, p.854–859.

13. **Lin F., (1993),** *"Robust and adaptive supervisory control of discrete event systems",* IEEE Transactions on Automatic Control, vol.38, no.12, p.1848–1852.

14. **Cury J.E.R., Krogh B.H., (1999) ,** *"Robustness of supervisors for discrete-event systems",* IEEE Transactions on Automatic Control, vol.44, no.2, p.376–379.

15. **Takai S., (2000)**, *"Robust supervisory control of a class of timed discrete event systems under partial observation",* Systems & Control Letters, vol.39, no.4, p.267–273.

16. **Bourdon S.E., Lawford M., Wonham W.M., (2005)**, *"Robust nonblocking supervisory control of discrete-event systems",* IEEE Transactions on Automatic Control, vol.50, no.12, p.2015–2021.

17. **Saboori A., Zad S.H., (2006),** *"Robust nonblocking supervisory control of discrete-event systems under partial observation",* Systems &Control Letters, vol.55, no.10, p.839 – 848.

18. **Park S.J., Lim J.T., (1998)** , *"Robust and fault-tolerant supervisory control of discrete event systems with partial observation and model uncertainty",* International Journal of Systems Science, vol.29, no.9, p.953–957.

19. **Wen Q., Kumar R. , Huang J., Liu H., (2008),** *" A framework for fault-tolerant control of discrete event systems",* IEEE Transactions on Automatic Control, vol.53, no.8, p.1839–1849.

20. **Wen Q., Kumar R., Huang J., (2008),** *"Synthesis of optimal fault-tolerant supervisor for discrete event systems"*, American Control Conference, Seattle, p.1172 –1177.

21. **Moor T., Schmidt K. W., (2015),** *"Fault-tolerant control of discrete-event systems with lower-bound specifications"*, Workshop on Dependable Control of Discrete Systems, Cancun.

22. **Ramadge P. J., Wonham W. M., (1987)**, *"Supervisory control of a class of discrete event processes"*, SIAM J. Control Optim, vol.25, no.1, p.206–230.

23. **libFAUDES, (2006–2013),** *"libFAUDES software library for discrete event systems"*.

APPENDIX

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: ACAR, Ayşe Nur
Date and Place of Birth: 25 April 1989, Ankara
Marital Status: Married
Phone: +90 507 869 29 55
Email: anursulek@cankaya.edu.tr

EDUCATION

Degree	Institution	Year
M.Sc.	Çankaya University, Electronic and Communication Engineering	2015
B.Sc.	Çankaya University, Electronic and Communication Engineering	2012
High School	Çankaya Milli Piyango Anatolian High School	2007

FOREIGN LANGUAGES

Turkish, English.

PUBLICATIONS

Sülek, A.N., Afşar, K. U., Schmidt, K. W.: Yeniden Yapılandırılabilir Üretim Sistemleri için Kontrolcü Tasarımı, Cankaya University Journal of Science and Engineering, 2013.

Acar, A.N., Schmidt, K.W.: Discrete Event Supervisor Design and Application for Manufacturing Systems with Arbitrary Faults and Repairs, IEEE International Conference on Automation Science and Engineering, Gothenburg, Sweden, 2015.

Sülek, A.N., Schmidt, K.W.: Computation of Supervisors for Fault-Recovery and Repair for Discrete Event Systems, Workshop on Discrete Event Systems, Paris, France, 2014.

Sülek, A.N., Schmidt, K.W.: Computation of Fault-Tolerant Supervisors for Discrete Event Systems, Workshop on Dependable Control of Discrete Systems, York, England, 2013.

Sülek, A., Afşar, K. U., Schmidt, K. W.: Yeniden Yapılandırılabilir Üretim Sistemleri için Kontrolcü Tasarimi, Mühendislik ve Teknoloji Semposyumu, Ankara, Türkiye, 2012.