Analysis and Control for Resilience of Discrete Event Systems

Fault Diagnosis, Opacity and Cyber Security

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

Other titles in Foundations and Trends® in Systems and Control

Learning-Based Control: A Tutorial and Some Recent Results
Zhong-Ping Jiang, Tao Bian and Weinan Gao
ISBN: 978-1-68083-752-0

Synchronous Reinforcement Learning-Based Control for Cognitive Autonomy
Kyriakos G. Vamvoudakis and Nick-Marios T. Kokolakis
ISBN: 978-1-68083-744-5

Achieving Ecological Resilience Through Regime Shift Management
M. D. Lemmon
ISBN: 978-1-68083-716-2

Distributed Optimization for Smart Cyber-Physical Networks
Giuseppe Notarstefano, Ivano Notarnicola and Andrea Camisa
ISBN: 978-1-68083-618-9

Resilient Control in Cyber-Physical Systems: Countering Uncertainty, Constraints, and Adversarial Behavior
Sean Weerakkody, Omur Ozel, Yilin Mo and Bruno Sinopoli
ISBN: 978-1-68083-586-1

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

Analysis and Control for Resilience of Discrete Event Systems

Fault Diagnosis, Opacity and Cyber Security

João Carlos Basilio
Universidade Federal do Rio de Janeiro
basilio@dee.ufrj.br

Christoforos N. Hadjicostis
University of Cyprus
chadjic@ucy.ac.cy

Rong Su
Nanyang Technological University
rsu@ntu.edu.sg

now
the essence of knowledge
Boston — Delft

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

Foundations and Trends® in Systems and Control

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

J. Basilio, C. Hadjicostis and R. Su. *Analysis and Control for Resilience of Discrete Event Systems*. Foundations and Trends® in Systems and Control, vol. 8, no. 4, pp. 285–443, 2021.

ISBN: 978-1-68083-857-2
© 2021 J. Basilio, C. Hadjicostis and R. Su

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without prior written permission of the publishers.

Photocopying. In the USA: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by now Publishers Inc for users registered with the Copyright Clearance Center (CCC). The 'services' for users can be found on the internet at: www.copyright.com

For those organizations that have been granted a photocopy license, a separate system of payment has been arranged. Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to now Publishers Inc., PO Box 1024, Hanover, MA 02339, USA; Tel. +1 781 871 0245; www.nowpublishers.com; sales@nowpublishers.com

now Publishers Inc. has an exclusive license to publish this material worldwide. Permission to use this content must be obtained from the copyright license holder. Please apply to now Publishers, PO Box 179, 2600 AD Delft, The Netherlands, www.nowpublishers.com; e-mail: sales@nowpublishers.com

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

Foundations and Trends® in Systems and Control
Volume 8, Issue 4, 2021

Editorial Board

Editors-in-Chief

Panos J. Antsaklis
University of Notre Dame
United States

Alessandro Astolfi
Imperial College London, United Kingdom
University of Rome "Tor Vergata", Italy

Editors

John Baillieul
Boston University

Peter Caines
McGill University

Christos Cassandras
Boston University

Denis Dochain
UC Louvain

Magnus Egerstedt
Georgia Institute of Technology

Karl Henrik Johansson
KTH Stockholm

Miroslav Krstic
University of California, San Diego

Jan Maciejowski
University of Cambridge

Dragan Nesic
The University of Melbourne

Marios Polycarpou
University of Cyprus

Jörg Raisch
Technical University Berlin

Arjan van der Schaft
University of Groningen

M. Elena Valcher
University of Padova

Richard Vinter
Imperial College London

George Weiss
Tel Aviv University

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

Editorial Scope

Topics

Foundations and Trends® in Systems and Control publishes survey and tutorial articles in the following topics:

- Control of:
  - Hybrid and Discrete Event Systems
  - Nonlinear Systems
  - Network Systems
  - Stochastic Systems
  - Multi-agent Systems
  - Distributed Parameter Systems
  - Delay Systems
- Filtering, Estimation, Identification
- Optimal Control
- Systems Theory
- Control Applications

Information for Librarians

Foundations and Trends® in Systems and Control, 2021, Volume 8, 4 issues. ISSN paper version 2325-6818. ISSN online version 2325-6826. Also available as a combined paper and online subscription.

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

Contents

1 Introduction and Motivation **3**

2 Notation and Preliminaries **10**
2.1 Introduction 10
2.2 Languages and Finite Automata 10
2.3 Observation Models and Observer Constructions 13
2.4 Supervisory Control 15

3 Fault Diagnosis and Safe Diagnosability **21**
3.1 Introduction 21
3.2 Diagnosability and Online Diagnosis 22
3.3 Diagnosability Verification 29
3.4 Robust Diagnosis of DES Subject to Int. Loss of Observations 39
3.5 Safe Diagnosability in Fault Tolerant Control 48
3.6 Diagnosability of Networked DES 60
3.7 Concluding Remarks and Future Research Directions 72

4 Opacity, Verification and Enforcement **74**
4.1 Introduction 74
4.2 Notions of Opacity 77
4.3 Opacity Verification 84
4.4 Opacity Enforcement 87
4.5 Concluding Remarks and Future Research Directions 101

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

5 Cyber Security in Discrete Event Systems **103**
5.1 Introduction 103
5.2 Sensor Attacks 106
5.3 Actuator Attacks 114
5.4 Sensor+Actuator Attacks 123
5.5 Concluding Remarks and Future Research Directions 138

6 Conclusions **141**

Acknowledgements **144**

References **145**

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

Analysis and Control for Resilience of Discrete Event Systems

João Carlos Basilio[1], Christoforos N. Hadjicostis[2] and Rong Su[3]

[1] *Universidade Federal do Rio de Janeiro; basilio@dee.ufrj.br*
[2] *University of Cyprus; chadjic@ucy.ac.cy*
[3] *Nanyang Technological University; rsu@ntu.edu.sg*

ABSTRACT

System resilience captures the ability of the system to withstand a major disruption within acceptable performance degradation and to recover within an acceptable time frame. In this monograph we consider two possible sources of major disruptions, i.e., *component faults* and *cyber intrusions*. A component fault is an indigenous activity that renders unavailability or inaccessibility of certain functions within a component, either permanently or temporarily. It typically generates safety and performance concerns. Cyber intrusion on the other hand is an exogenous activity that tampers privacy, confidentiality, availability, or integrity of the system. These two sources are not always independent from each other. For example, a cyber intrusion may trigger a component fault, whereas a component fault may open a door for cyber intrusion, e.g., by keeping it undetected. For cyber intrusion, we will focus on *opacity*, which describes the system's ability to hide certain secrets from an external observer (or eavesdropper), and *sensor and actuator attacks* that exploit the system's existing controller to generate undesirable behaviours.

João Carlos Basilio, Christoforos N. Hadjicostis and Rong Su (2021), "Analysis and Control for Resilience of Discrete Event Systems", Foundations and Trends® in Systems and Control: Vol. 8, No. 4, pp 285–443. DOI: 10.1561/2600000024.

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

In this monograph, we provide a detailed account of most recent research outcomes on *fault diagnosis*, *opacity analysis and enhancement*, and *cyber security analysis and enforcement*, within suitable discrete event system modelling frameworks. In each case, we describe basic problem statements and key concepts, and then point out the key challenges in each research area. After that, we present a thorough review of state-of-the-art techniques, and discuss their advantages and disadvantages. Finally, we highlight key research directions for further exploration.

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

1

Introduction and Motivation

System resilience refers to a system's ability to withstand possibly disruptive changes and maintain acceptable performance degradation. A more control-oriented definition can be found in Rieger *et al.* (2009), which states that *a resilient control system is one that maintains state awareness and an accepted level of operational normalcy in response to disturbances, including threats of an unexpected and malicious nature.* For example, when component failures take place, resilience describes how quickly relevant faults can be identified and whether the system can still run but possibly with degraded performance. When a cyber attack occurs, resilience describes how quickly the system controller can detect the presence of an attack, and whether a counter measure can be taken to mitigate the impact of the attack. When there is an eavesdropper, resilience describes whether it is still possible for the system controller to ensure key information remains confidential. When privacy is concerned, resilience describes whether a certain type of public interference can be effectively stopped. With the fast growing adoption of the cyber physical system (CPS) and internet of things (IoT) concepts, resilience has been playing an increasingly important role in measuring a system's ability to ensure information confidentiality, personal privacy, service availability and data integrity.

There has been a large volume of research on system resilience. Some recent surveys can be found in, e.g., Dibaji *et al.* (2019) on systems and

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

control perspectives of CPS, Hosseini *et al.* (2016) on definitions and measures of resilience, Uday and Marais (2015) on metrics, methods and challenges in designing resilient systems-of-systems, Jang-Jaccard and Nepal (2014) on security vulnerabilities and mitigation techniques, and new cyber attack patterns in information infrastructures, and Knowles *et al.* (2015) on cyber security management in industrial control systems. Most systems under consideration are time driven. Nevertheless, many man-made systems exhibit event-driven characteristics, which cannot be properly treated by time-driven models.

In this monograph we will focus on event-driven systems, or the so-called *discrete event systems* (DES). In a DES, state changes are triggered by events, instead of time – the latter in the discrete case may also be considered as events. States are typically discrete, and transitions are asynchronous, i.e., up to one event firing at each time, and nondeterministic, i.e., at each state there may be more than one event choice. There are several DES modelling frameworks such as, regular languages and finite automata, Petri nets, queuing models, Markov processes, process algebra, and max-plus algebra. DES models can be used to describe many industrial systems, e.g., manufacturing systems, logistic systems, medicare systems, robotic systems, transportation systems and telecommunication systems. A DES model essentially (exhaustively) specifies sequential orders of individual functions captured by events. For example, it is rather straightforward to specify which event (or function) should take place before another event (or function); which event or sequence of events should always or never take place; and which state should be reached eventually or never be reached. When equipped with suitable composition rules, DES models are amenable for formal treatments such as model checking, analysis and synthesis, making them perfect candidates to ensure correctness-by-design, which is highly sought after in industry. When states and transitions are equipped with numerical weights, a DES model is also suitable for performance optimization.

In this monograph we will focus on analysis and control for resilience of discrete event systems. Considering the broadness of this subject, we have decided to focus on component faults, system opacity and cyber attacks, which provide a good coverage over system's indigenous

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

activities (i.e., faults) and exogenous activities (i.e., eavesdropping and cyber attacks), and to address key system properties such as information confidentiality in opacity analysis, data integrity in cyber attack analysis, and service/function availability in cyber attack analysis and fault diagnosis. By presenting an intuitive description of basic problems in each area, followed by a detailed review of the state-of-the-art analysis and control techniques, we aim to illustrate key challenges ahead and suggest possible solution processes based on knowledge and experience gained from existing research outcomes.

Fault diagnosis has been one of the key research areas in the DES community for more than 20 years. Starting with the first well recognized state-based modeling framework introduced in Lin (1994), and an even more widely adopted language-based framework proposed in Sampath *et al.* (1995a), numerous research works have been published that address a large variety of fault scenarios. Basically, a fault is considered as a system condition that prevents the system from performing its normal operations or functions, whereas a failure refers to a set of symptoms that reveal functional deviations from those normally expected. In the DES fault diagnosis literature, a fault is usually modelled as an unobservable event. Fault diagnosis aims to detect symptoms in the system, and identify relevant faults that cause such symptoms. Fault detection is usually straightforward, as it is a binary decision process that decides whether there is any observable functional deviation (or abnormal activity) in the given system. The key challenge lies in the fault identification part, captured by the concept of diagnosability, which requires to determine the type of each (ongoing) fault that may explain those symptoms, after a finite number of observations are received. In this sense, fault diagnosability is directly related to the function availability property. A specific apparatus called diagnoser is usually constructed for fault detection and identification. There are several good historic accounts on fault diagnosis in the DES literature, e.g., (Zaytoon and Lafortune, 2013), (Lafortune *et al.*, 2018). In this monograph we will mainly focus on automaton-based (or regular language-based) frameworks, due to their conceptual simplicity. After introducing basic fault diagnosis problems and relevant verification algorithms, we describe a set of related topics that have recently received increased attention

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

from the research community. To address data generation and reception failures that may tamper the effectiveness of fault diagnosis, we discuss recent works on robust diagnosis, which take into account sensor and/or communication channel failures. We present, in the sequel, a review of research outcomes on fault-tolerant control, which is termed as safe diagnosis and aims for safe operation of a concerned supervisor, even when faults may occur; new verification algorithms for safe diagnosability verification are also presented. We end Chapter 3 addressing issues related to diagnosability of networked DES, which have recently become of great interest due to the growth in the use of communication networks in industrial systems.

Opacity is an information confidentiality property that is increasingly being used in various types of partially observed DES in order to systematically capture, verify and, if necessary, enforce important aspects of security and privacy in applications of such systems. Following its introduction less than two decades ago with the works in Bryans *et al.* (2004), Bryans *et al.* (2005b), Badouel *et al.* (2007), Saboori and Hadjicostis (2007), Bryans *et al.* (2008), opacity has emerged at the forefront of DES research. Interest in opacity analysis and enforcement is largely driven by technological developments and, in particular, the increasing reliance of numerous technological systems and applications (including banking systems, health databases, traffic networks, and power distribution systems) on shared cyber-infrastructures, such as the Internet or cellular networks. In this monograph we will review some of the most predominant formulations of opacity, along with methods for verifying them and enforcing them. In its most basic form, opacity aims to characterize the type of inferences and conclusions a passive observer can reach regarding a subset of the behaviour of a given DES. In a typical scenario, this subset of the behaviour of the system, which is referred to as the *secret behaviour* or simply the *secret* of the system, is captured by a predicate that can evaluate to either true or false. Under such a formulation, the fundamental question that opacity aims to determine whether it is possible for certain activity in the system to generate observations that will allow an external observer to assert the truth of the predicate. In this case, we say that "the secret is revealed" or that "opacity is violated." The external observer is usually assumed

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

passive, i.e., it can only obtain observations generated by system activity but cannot influence the system operation; however, the external observer may have partial knowledge of the system model, which it can utilize to determine (based on the observations) if the predicate is true. If opacity is violated, i.e., if there exists a system behaviour that allows the external observer to determine that secret behaviour has occurred, the next natural steps are to characterize the type of behaviour that leads to opacity violations, and enforce opacity by either (i) disallowing behaviour that violates opacity (e.g., using supervisory control) or (ii) by masking the output behaviour of the system (e.g., by obfuscating the observations that are generated by the system). Both approaches for opacity enforcement have their advantages and disadvantages. For instance, they both require suitable technological means, such as the ability to disable certain controllable events in the case of supervisory control and the ability to suppress certain system outputs (or even mask some of them to different outputs) in the case of obfuscation. Note that supervisory control will end up disabling system behaviour which may inhibit its functionality, whereas output obfuscation will need to ensure that a consistent behaviour that does not violate opacity is presented to the external observer (at all time instances, including future times). We elaborate on all of these issues later in Chapter 4.

Cyber-attack resilience refers to properties of service availability and data integrity. With the continuous advancement of information and communications technology (ICT), in particular, the recent 5G-based IoT technologies, we are enjoying unprecedented connectivity around the world. Nevertheless, the threat of cyber attacks that may potentially cause significant damage to human lives and properties has more frequently become the center of attention, and has been attracting lots of research from different communities. Basically, an attacker aims to inflict damage on a target system by disrupting its control loop. This could be achieved either by intercepting and changing the controller's input signals (in terms of sensor attacks), or by intercepting and changing the controller's output signals (in terms of actuator attacks), or by completely blocking the data transmission between the controller and the plant (in terms of denial-of-service attacks). An attack can be either brut-force, e.g., via hardware destruction or via signal jamming, or covert (or

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

stealthy), i.e., the attacking process is hidden from the controller or any other monitoring mechanism. A good survey of cyber attacks and cyber defence with a systems-and-control perspective can be found in Dibaji *et al.* (2019). Typically, linear systems are considered in existing works that rely on system identification and control techniques. Within the DES community, most works rely on the control system setup introduced in the Ramadge-Wonham supervisory control paradigm (Ramadge and Wonham, 1987), where the plant generates observable outputs, received by the supervisor via an observation channel, and each control command specified as a set of allowable events is generated by the supervisor and fed to the plant via a command channel. The plant will nondeterministically pick one event from the control command and execute it. Unlike attacks in time-driven systems described in Dibaji *et al.* (2019), attacks under consideration in a DES aim to change the order of events in specific system runs. There are two different streams of research on cyber attacks and resilient control. The first stream is to treat each attack as a fault that alters event orders in the system by either removing existing events or inserting fictitious events, see, e.g., (Carvalho *et al.*, 2016), (Carvalho *et al.*, 2018), (Lima *et al.*, 2018). In this case research focuses firstly on how to decide the existence of an attack in the system based on a finite number of observations, similar to verifying diagnosability in fault diagnosis, and secondly on how to ensure sufficient control abilities to stop the propagation of the attack effect, e.g., by shutting down the entire system. Partial observation and existence of uncontrollable events complicate the synthesis process. The second stream is to develop a specific "smart" attack model that ensures certain intuitive properties such as covertness and guaranteed (strong or weak) damage infliction, see, e.g., (Su, 2017), (Meira-Góes *et al.*, 2017), (Su, 2018), (Meira-Góes *et al.*, 2019), (Lin *et al.*, 2019a), (Zhu *et al.*, 2019). Under such a smart attack model, the research focuses on analyzing how the attack may affect the closed-loop behaviour, deciding whether the system is vulnerable with respect to such an attack, and finally computing a supervisor that is resilient to the concerned attack. In this monograph we will provide a detailed account of existing state-of-the-art cyber attack and resiliency results, and point out challenges for future exploration.

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

This monograph is organized as follows. Chapter 2 presents basic concepts and notations about discrete event systems. A detailed review of the state-of-the-art fault diagnosis techniques is provided in Chapter 3, followed by a review of recent works on opacity analysis and enforcement in Chapter 4. Chapter 5 gives a detailed account of recent research results on sensor and actuator attacks and attack-resilient supervisor synthesis. Conclusions are drawn in Chapter 6.

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

2

Notation and Preliminaries

2.1 Introduction

In this chapter, we introduce notations that will be used in the remainder of this monograph to facilitate discussions of fault diagnosis, opacity, and resilient control in discrete event systems (DES). Since the monograph focuses primarily on DES that can be modelled via regular languages and finite automata, we restrict attention to aspects pertaining to these models. However, the interested reader could pursue related extensions to other DES models, including Petri nets and probabilistic finite automata, as described in the relevant chapters that follow. More detailed discussions about the introductory material in this chapter can be found in Cassandras and Lafortune (2009), Hadjicostis (2020), Wonham and Cai (2019).

2.2 Languages and Finite Automata

Let Σ be a finite alphabet of events that can occur in a given DES. Each event in Σ denotes an *action* that is assumed to be instantaneous. Each finite sequence of events from Σ forms a finite *string*, including a special string ϵ, which denotes the empty string, whose length is considered

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

as 0. String $s = \sigma_{i_1}\sigma_{i_2}...\sigma_{i_n}$ can be *concatenated* with another string $t = \sigma_{j_1}\sigma_{j_2}...\sigma_{j_m}$, to form the string

$$st := \sigma_{i_1}\sigma_{i_2}...\sigma_{i_n}\sigma_{j_1}\sigma_{j_2}...\sigma_{j_m} ,$$

which captures the sequence of events in s followed by the sequence of events in t. We use Σ^* to denote the set of all finite strings, and for any $s \in \Sigma^*$ we have $s\epsilon = \epsilon s = s$. Formally speaking, Σ^* is the free monoid over Σ, whose unit element is ϵ and the binary operation involved is concatenation. Given two strings $s, t \in \Sigma^*$, we say s is a *prefix substring* of t, denoted as $s \leq t$, if there exists $u \in \Sigma^*$ such that $su = t$. We use $\|s\|$ to denote the length of s; by convention, $\|\epsilon\| = 0$.

A *language* $L \subseteq \Sigma^*$ is a subset of finite-length strings in Σ^*. The *prefix closure* of L is defined as $\bar{L} := \{s \in \Sigma^* | (\exists t \in L) s \leq t\}$. If $L = \bar{L}$, then L is called *prefix closed*. Given two languages L_1 and L_2, one can talk about the union of the two languages $L_1 \cup L_2$ or the intersection of the two languages $L_1 \cap L_2$ in the manner associated with set operations:

$$\begin{aligned} L_1 \cup L_2 &= \{s \in \Sigma^* | s \in L_1 \text{ or } s \in L_2\} , \\ L_1 \cap L_2 &= \{s \in \Sigma^* | s \in L_1 \text{ and } s \in L_2\} . \end{aligned}$$

For notational simplicity, we will use "$\wedge$" to denote the logical conjunction "AND", and "$\vee$" for logical disjunction "OR". In addition, we use $L_1L_2 := \{st \in \Sigma^* | s \in L_1 \wedge t \in L_2\}$ to denote the concatenation of two languages. We use $|A|$ to denote the size (i.e., the cardinality or number of elements) of an arbitrary set A.

We also use standard regular expression notations to describe a set of strings from Σ^* that follow a particular pattern. More specifically, we make use of the following operators:

1. The superscript ^{n} (where n is a positive integer) is used to denote n repetitions of its argument. For example, given a string s, s^3 denotes the singleton set $\{sss\}$ or sometimes, abusing notation, the string sss itself. Similarly, ab^nc denotes the singleton set $\{abb..bc\}$ (or string $abb..bc$) where b appears n times.

2. Given a string s, s^* denotes the set of strings that involve an arbitrary (but finite) number of repetitions of s, including no repetition, i.e., $s^* = \{\epsilon, s, ss, sss, ...\}$. Similarly, ab^*c denotes the set of strings that involve an a, followed by zero, one or more occurrences of b, followed by c, i.e., $ab^*c = \{ac, abc, abbc, abbbc, ...\}$.

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

3. The + operator is used to denote unions of sets of strings. For example, $a^* + bc^*d$ denotes the set of strings of the form $aa...a$ (where a appears zero, one or more times) or b followed by zero, one or more occurrences of c, followed by d. In other words, $a^* + bc^*d = \{\epsilon, a, aa, aaa, ..., bd, bcd, bccd, ...\}$. The above operations can be combined. For example, $(a+b)^*c(d+e)^*$ denotes the following set of strings (ordered according to their length):

$$\{c, ac, bc, cd, ce, acd, ace, bcd, bce, aac, bbc, abc, bac, cdd, cee, cde, ced, ...\} .$$

Definition 2.1 (Nondeterministic Finite Automaton (NFA))**.** A non-deterministic finite automaton is captured by $G_n = (X, \Sigma, \delta, X_0, X_m)$, where $X = \{x_1, x_2, \ldots, x_N\}$ is the finite set of states, Σ is the finite alphabet, $\delta : X \times \Sigma \to 2^X$ is the nondeterministic state transition function, $X_0 \subseteq X$ is the set of possible initial states, and $X_m \subseteq X$ is the set of marker (or final) states. When $X = X_m$, we may simply write $G_n = (X, \Sigma, \delta, X_0)$. □

Given that the NFA G_n is in a particular state $x \in X$, the occurrence of event $\sigma \in \Sigma$ causes a transition to a state in the set $\delta(x, \sigma)$. For each subset of states $X' \subseteq X$ and each $\sigma \in \Sigma$, we can extend the definition of δ as

$$\delta(X', \sigma) := \cup_{x' \in X'} \delta(x', \sigma) .$$

With this notation at hand, we can also extend δ to the domain $X \times \Sigma^*$ in the following manner: for all $x \in X$, all $\sigma \in \Sigma$, and all $s \in \Sigma^*$, we define

$$\delta(x, s\sigma) := \delta(\delta(x, s), \sigma) ,$$

where the base case is $\delta(x, \epsilon) := \{x\}$.

The language *generated* by NFA G_n (or the *closed behaviour* of G_n) captures all allowable sequences of events and is defined as

$$L(G_n) := \{s \in \Sigma^* | (\exists x_0 \in X_0)\, \delta(x_0, s) \neq \varnothing\}.$$

The language *recognized* by G_n (or the *marked behaviour* of G_n) is defined as

$$L_m(G_n) := \{s \in L(G_n) | \delta(X_0, s) \cap X_m \neq \varnothing\}.$$

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

Later, we will interpret each string $s \in L(G_n)$ as a task, either completed or uncompleted, and each string $t \in L_m(G_n)$ will denote a completed task.

If we let $L(G_n, x)$ denote the set of all traces that originate from state x of G_n (i.e., $L(G_n, x) := \{s \in \Sigma^* | \delta(x, s) \neq \emptyset\}$), then we have $L(G_n) = \bigcup_{x_0 \in X_0} L(G_n, x_0)$.

When $|X_0| = 1$, i.e., $X_0 = \{x_0\}$ for some $x_0 \in X$, and for all $x \in X$ and all $\sigma \in \Sigma$, we have $|\delta(x, \sigma)| \leq 1$, we call G_n a deterministic finite automaton (DFA). When $|\delta(x, \sigma)| = 0$ we say that $\delta(x, \sigma)$ is undefined. By convention, the original transition map δ for an NFA becomes a (partial) transition map $\delta : X \times \Sigma \to X$, which may not be defined for some pairs in $X \times \Sigma$. To avoid any unnecessary confusion, we use $G_d = (X, \Sigma, \delta, x_0, X_m)$ (or $G = (X, \Sigma, \delta, x_0, X_m)$) to denote a DFA. The function δ of DFA G_d can be extended from the domain $X \times \Sigma$ to the domain $X \times \Sigma^*$ as follows: for all $x \in X$, all $\sigma \in \Sigma$, and all $s \in \Sigma^*$, we define

$$\delta(x, s\sigma) := \delta(\delta(x, s), \sigma) \ ,$$

where $\delta(x, \epsilon) := x$, and $\delta(x, s\sigma)$ is undefined if $\delta(x, s)$ is undefined. Again, the behaviour of G_d is captured by its languages $L(G_d)$ and $L_m(G_d)$. As before, $L(G_d, x)$ denotes the set of all traces that originate from state x of G_d.

2.3 Observation Models and Observer Constructions

The most common way of capturing the capability of the external observer of a given NFA is to assume that it can only observe a subset Σ_o ($\Sigma_o \subseteq \Sigma$) of the events. The remaining events are unobservable and are denoted by $\Sigma_{uo} := \Sigma - \Sigma_o$. The natural projection $P_o : \Sigma^* \to \Sigma_o^*$ can be used to map any string executed in the system to the sequence of observations associated with it. The natural projection is defined recursively as $P_o(s\sigma) = P_o(s)P_o(\sigma)$, for all $s \in \Sigma^*$ and $\sigma \in \Sigma$, with

$$P_o(\sigma) = \begin{cases} \sigma, & \text{if} \quad \sigma \in \Sigma_o, \\ \epsilon, & \text{if} \quad \sigma \in \Sigma_{uo} \cup \{\epsilon\}, \end{cases}$$

where ϵ represents the empty (observation) trace. Given a language $L \subseteq \Sigma^*$, we define $P_o(L) := \{P_o(s) | s \in L\}$. We also define the inverse

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

image map of P_o as $P_o^{-1} : \Sigma_o^* \to 2^{\Sigma^*}$, where

$$(\forall t \in \Sigma_o^*)\, P_o^{-1}(t) := \{s \in \Sigma^* | P_o(s) = t\}.$$

Given a language $L \subseteq \Sigma_o^*$, we define $P_o^{-1}(L) := \cup_{s \in L} P_o^{-1}(s)$.

Given an NFA $G_n = (X, \Sigma, \delta, X_0)$ (or a DFA $G_d = (X, \Sigma, \delta, X_0)$) in which the sequence of events s has occurred, the observer records the sequence of observations $\omega = P_o(s)$. Based on the sequence of observations ω, the observer might be interested in a variety of estimation tasks, including current state estimation and initial state estimation. Specifically, the set of possible current states for system G_n given the sequence of observations ω is captured by

$$\hat{X}(\omega) = \{x \in X | (\exists s \in \Sigma^*)(\exists x_0 \in X_0)\, x \in \delta(x_0, s) \wedge P_o(s) = \omega\}.$$

Similarly, the set of possible initial states for system G_n given the sequence of observations ω is given by

$$\hat{X}_0(\omega) = \{x_0 \in X_0 | (\exists s \in \Sigma^*)\, \delta(x_0, s) \neq \varnothing \wedge P_o(s) = \omega\}\ .$$

The interested reader may find details about how to systematically obtain estimator constructions for current and initial state estimation as well as fault diagnosis in Cassandras and Lafortune (2009) and Hadjicostis (2020). Some relevant discussions are also included later in this monograph. In particular, the observer or current-state estimator for a given NFA G_n is described below.

Definition 2.2 (Observer or Current-State Estimator)**.** Consider an NFA $G_n = (X, \Sigma, \delta, X_0)$ under a natural projection map P_o with respect to a set of observable events Σ_o, $\Sigma_o \subseteq \Sigma$. Its observer automaton $G_{obs} = (X_{obs}, \Sigma_o, \delta_{obs}, X_{0,obs})$ is a deterministic finite automaton where:
1. $X_{obs} \subseteq 2^X$ which means that the states of G_{obs} are subsets of X.
2. The initial state is given by

$$X_{0,obs} = \{x \in X \mid (\exists t \in \Sigma_{uo}^*) x \in \delta(X_0, t)\}\ ,$$

i.e., $X_{0,obs}$ is the set of states of G_n that can be reached from an initial state in X_0 via zero, one, or more unobservable events.
3. For $x_{obs} \in X_{obs}$ and $\sigma_o \in \Sigma_o$, the next state transition function δ_{obs} is defined as $x'_{obs} = \delta_{obs}(x_{obs}, \sigma_o)$ with

$$x'_{obs} = \{x' \in X | (\exists s \in \Sigma^*) x' \in \delta(x_{obs}, s) \wedge P_o(s) = \sigma_o\}\ .$$

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

We typically limit X_{obs} to be the set of states in G_{obs} that are reachable from its initial state $X_{0,obs}$. Also, we do not include x'_{obs} in the construction of G_{obs} if x'_{obs} is the empty set (i.e., if $\delta(x_{obs}, \sigma_o) = \varnothing$, we take $\delta_{obs}(x_{obs}, \sigma_o)$ to be undefined).

It is not hard to verify that the observer automaton has the following fundamental property (Cassandras and Lafortune, 2009), (Hadjicostis, 2020): given any sequence of events $s \in L(G_n)$ that occurs in system G_n and generates the sequence of observations $\omega = P_o(s)$, $\omega \in \Sigma_o^*$, we have

$$\hat{X}(\omega) = \delta_{obs}(X_{0,obs}, \omega) \ , \tag{2.1}$$

i.e., the set of possible current states of G_n given the observation of the sequence ω is captured by the (unique) state reached in the observer starting from the initial state and applying the sequence ω.

2.4 Supervisory Control

In this section we introduce basic concepts of the Ramadge-Wonham supervisory control paradigm (Ramadge and Wonham, 1987). A given plant, typically modelled as a DFA $G = (X, \Sigma, \delta, x_0, X_m)$, constantly generates event outputs that may be externally observed. The plant alphabet Σ is partitioned into *controllable* and *uncontrollable* alphabets, represented by Σ_c and Σ_{uc}, respectively. Each controllable event $\sigma \in \Sigma_c$ denotes one plant action (or function) that can be disabled externally, typically by a controller, whereas each uncontrollable event $\sigma' \in \Sigma_{uc}$ denotes an action (or function) whose occurrence cannot be stopped by the controller. Some examples of controllable actions include turning on or off a machine in manufacturing, opening or closing a door in access control, or dispatching or retrieving AGVs in robotic operations. Some examples of uncontrollable actions include component failures, completion of a chemical reaction, sensor output values, or cyber attacks, which are beyond the reach of the controller. To capture observability of the plant output, the alphabet Σ is also partitioned into observable and unobservable alphabets, represented by Σ_o and Σ_{uo}, respectively. Each observable event is typically associated with an action (or function) whose execution can be monitored by sensors, whereas an unobservable

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

event denotes an action (or function) whose execution cannot be detected externally by the controller. All observable output sequences are captured by the language $P_o(L(G))$, where P_o is the natural projection described in the previous section.

A supervisor is modelled as a map

$$V : P_o(L(G)) \to \Gamma := \{\gamma \subseteq \Sigma : \Sigma_{uc} \subseteq \gamma\},$$

where Γ is the set of all *control patterns* (or *control commands*), and each control pattern $\gamma \in \Gamma$ must contain all uncontrollable events, i.e., $\Sigma_{uc} \subseteq \gamma$. In the Ramadge-Wonham paradigm, the supervisor sends a control pattern to the plant G, instead of a single event typically seen in the systems and control community. This is mainly due to the existence of uncontrollable events, which does not allow the supervisor to generate a unique control input to the plant G. By adopting this nondeterministic execution mechanism, the plant G is expected to choose one event from the control pattern and execute it. The details of this nondeterministic choice are not explicitly explained in the original Ramadge-Wonham paradigm, but could play a key role in the current networked control frameworks with imperfect communication channels, and cyber security analysis and control, which will be discussed in Chapter 5.

The standard Ramadge-Wonham supervisory control architecture is depicted in Fig. 2.1, where event executions in the plant G are

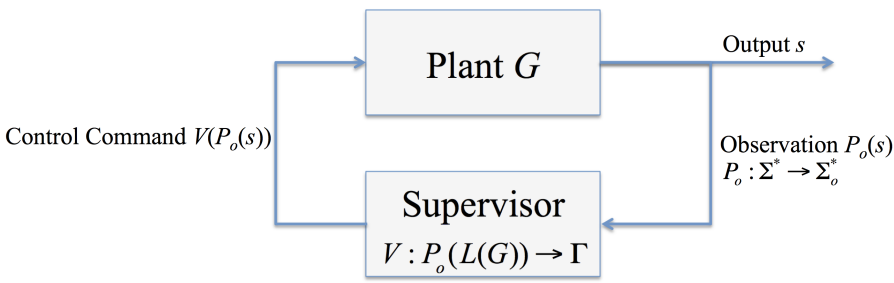


Figure 2.1: The Ramadge-Wonham supervisory control architecture.

assumed to be instantaneous and asynchronous, i.e., at any time instant there can be no more than one event being executed. Although these assumptions have been relaxed in works on time-weighted automata and

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

product systems that allow non-zero durations for event occurrences and concurrent event occurrences, respectively, in this monograph we keep these assumptions to make relevant analyses and synthesis simple and easily comprehensible.

If we carefully look at the definition of the supervisory control map V, we can see that the definition imposes two constraints on the supervisor. That is, first of all, the supervisor cannot disable any uncontrollable events, which is reflected in the property that $\Sigma_{uc} \subseteq \gamma$ for each control pattern $\gamma \in \Gamma$, and secondly, if two strings $s, s' \in L(G)$ are observably identical, i.e., $P_o(s) = P_o(s')$, then the resulting control patterns must be the same, which is reflected in the domain of V that is defined over $P_o(L(G))$. With the influence of V, the closed-loop system is denoted as V/G, which is treated as another DFA. The closed behaviour of V/G is defined as follows:

1. $\epsilon \in L(V/G)$,
2. $(\forall s \in L(V/G))(\forall \sigma \in \Sigma)[s\sigma \in L(G) \wedge \sigma \in V(P_o(s)) \Rightarrow s\sigma \in L(V/G)]$,
3. All strings in $L(V/G)$ are generated in Steps (1)–(2).

The marked behaviour of V/G is defined as follows:

$$L_m(V/G) = L(V/G) \cap L_m(G).$$

In the case that $L(V/G)$ is regular (i.e., it is a language that can be generated by a finite automaton), there exists a DFA $S = (Y, \Sigma, \eta, y_0, Y_m)$ such that

$$L(V/G) = L(S) \cap L(G) \text{ and } L_m(V/G) = L_m(S) \cap L_m(G).$$

Fig. 2.2 depicts the realization of V by automaton S, where the output control pattern of S for each observation input $P_o(s) \in P_o(L(G))$ is simply the collection of all outgoing transitions after $P_o(s)$ in S, i.e., $En_S(P_o(s)) := \{\sigma \in \Sigma : P_o(s)\sigma \in L(S)\}$.

There are two fundamental questions in the Ramadge-Wonham paradigm, which are listed below:

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

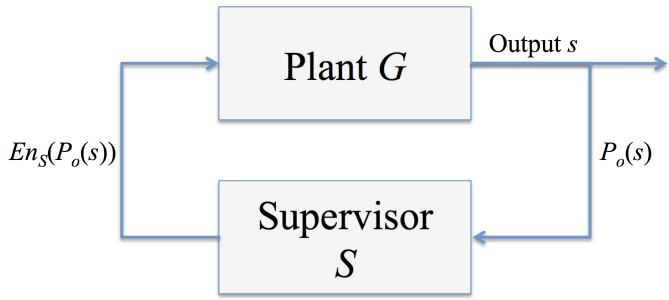


Figure 2.2: The automaton realization of V.

Problem 1. What language $K \subseteq L_m(G)$ admits a supervisor V such that $K = L_m(V/G)$?

Problem 2. How can one synthesize a supervisor V such that $L_m(V/G) \subseteq E \subseteq L_m(G)$, where E is a requirement language, and the closed-loop system is *nonblocking* (to ensure that every incomplete task may be completed later), i.e., $L(V/G) = \overline{L_m(V/G)}$?

To answer the first question, two fundamental concepts are introduced, i.e., controllability and observability.

Definition 2.3 (Language Controllability)**.** A language $K \subseteq L_m(G)$ is *controllable* with respect to G and Σ_{uc}, if $\bar{K}\Sigma_{uc} \cap L(G) \subseteq \bar{K}$. □

Definition 2.4 (Language Observability)**.** (Lin and Wonham, 1988) A language $K \subseteq L(G)$ is *observable* with respect to G and P_o, if

$$(\forall s, s' \in \overline{K})(\forall \sigma \in \Sigma)\,[s\sigma \in \overline{K} \wedge s'\sigma \in L(G) \wedge P_o(s) = P_o(s') \Rightarrow s'\sigma \in \overline{K}].$$

A language $K \subseteq L(G)$ is (prefix) *normal* with respect to G and P_o, if

$$P_o^{-1}(P_o(\overline{K})) \cap L(G) = \overline{K}.$$

A language $K \subseteq C \subseteq L(G)$ is *relatively observable* (Wonham and Cai, 2019) with respect to $\overline{C}$, G and P_o, or simply $\overline{C}$*-observable*, if for all $s \in \overline{K}$, $s' \in \overline{C}$ and $\sigma \in \Sigma$,

$$[s\sigma \in \overline{K} \wedge s'\sigma \in L(G) \wedge P_o(s) = P_o(s')] \Rightarrow s'\sigma \in \overline{K}.$$

□

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

It has been shown that (prefix) normality implies observability, and observability together with the condition that $\Sigma_c \subseteq \Sigma_o$ implies (prefix) normality. When setting $C = K$, $\overline{C}$-observability becomes observability, which suggests that $\overline{C}$-observability implies observability. On the other hand, when K is (prefix) normal with respect to G and P_o, then K is also $\overline{C}$-observable, namely (prefix) normality implies $\overline{C}$-observability. With the concepts of language controllability and language observability, a sufficient and necessary condition to ensure the existence of a supervisory control map for a given language $K \subseteq L_m(G)$ is given in the following theorem.

Theorem 2.1. A language $K \subseteq L_m(G)$ is realizable by a supervisory control map V, i.e., $K = L_m(V/G)$, if and only if K is controllable with respect to G and Σ_{uc}, observable with respect to G and P_o, and $L_m(G)$-closed, i.e., $\bar{K} \cap L_m(G) = K$. □

Intuitively, controllability of K ensures the existence of a supervisor V that does not disable any uncontrollable event, and observability of K ensures that V can be defined over $P_o(L(G))$. The last condition of being $L_m(G)$-closed is to ensure that any string $s \in K$ can be identified when it occurs. Since the supervisor V does not carry any marking information, to recognize a string in K, we can only rely on the marking of the plant G, i.e., we need to ensure that, under the influence of V, a string $s \in K$ holds if and only if $s \in L(V/G)$ and $s \in L_m(G)$.

To answer the second question, we need to find a controllable, observable and nonblocking sublanguage of E. A solution is typically not unique. This may cause practical issues, as sometimes trivial solutions may be generated. To avoid this unpleasant situation, it has been shown that, controllable and observable languages form a partially ordered set (poset) with set inclusion as the partial order. Thus, typically a maximal controllable and observable sublanguage within a given requirement language E is sought in synthesis. When observability is strengthened as relative observability or normality, the poset becomes an upper semilattice and a unique supremal element exists, which is sought in most supervisor synthesis literature. Tools are available to synthesize such supremal controllable and normal (or conditionally observable) sublanguages, e.g., see SuSYNA developed at Nanyang Technological University

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

(https://www.ntu.edu.sg/home/rsu/Downloads.htm), TCT developed at the University of Toronto (https://www.control.utoronto.ca/~wonham/Research.html), and Supremica developed at Chalmers University of Technology (https://supremica.org/).

State-based supervisory control can be used to solve the state avoidance control problem as described below. One should keep in mind that both language- and state-based frameworks for supervisory control are equivalent via appropriate transformation.

Definition 2.5 (State Avoidance Control Problem.)**.** Consider a DFA $G = (X, \Sigma, \delta, x_0, X_m)$, where Σ_c, $\Sigma_c \subseteq \Sigma$ is the set of controllable events and $\Sigma_{uc} = \Sigma - \Sigma_c$ is the set of uncontrollable events. A supervisor S is given by a function

$$S : X \to \Gamma := \{\gamma \subseteq \Sigma : \Sigma_{uc} \subseteq \gamma\} ,$$

which specifies the set of events that are enabled at state x, $x \in X$, by the controller. We write S/G for the closed-loop system, consisting of DFA G controlled by the supervisor S. The state avoidance control problem computes a maximally permissive supervisor while ensuring that states in a given set of forbidden states are not visited in the controlled system.

This section is not meant to provide a thorough overview of supervisory control theory, which would fit in a separate monograph. More advanced supervisory control theory, e.g., decentralized control, modular control, distributed control, hierarchical control, symbolic synthesis, etc., can be found in (Cassandras and Lafortune, 2009), (Wonham and Cai, 2019).

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

3

Fault Diagnosis and Safe Diagnosability

3.1 Introduction

Although the use of Petri nets as a modeling formalism in fault diagnosis goes back to early 1970 decade, it is well accepted that the problem of fault diagnosis first appeared in the DES community in Lin (1994), who proposed a formulation using a state-based approach. Not much later, the most widely accepted formulation to the problem of fault diagnosis of DES, the so-called event sequence approach, was introduced (Sampath *et al.*, 1995a). According to Sampath *et al.* (1995a), the fault is an unobservable event whose occurrence must be detected based solely on the observation of sequences of observable events. In addition, such a detection must be performed in finite time, i.e., within the occurrence of a finite number of events after the fault occurrence. It is worth remarking that these two works use automaton as the modeling formalism, whereas the first works that use both 'Petri net' and 'discrete event system' expressions in the context of fault diagnosis only appeared much later in 2003 (Genc and Lafortune, 2003), (Boel and Jiroveanu, 2003), (Alcaraz-Mejia *et al.*, 2003). Due to its close relation to practical problems, fault diagnosis of DES has become an active area of research within the DES community since its formulation in 1994 (see Zaytoon and Lafortune (2013), Lafortune *et al.* (2018)).

Documents
Document Sharing Hub

Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

We consider in this chapter the problem of fault diagnosis of DES using automata as the underlying modeling formalism. We start by presenting in Section 3.2 the formal definition of language diagnosability, and, after that, we discuss how fault diagnosis can be performed online; we will refer to the apparatus used to perform fault diagnosis as the diagnoser. We then present in Section 3.3 efficient ways to perform diagnosability verification. In order to keep the presentation simple, we restrict ourselves to monolithic systems. It is worth remarking that the DES diagnosis considered up to and including Section 3.3 relies on the accurate observation of all observable event occurrences. Since event occurrence is recorded by sensors, diagnosers are unlikely to work properly when sensor signals do not reach the diagnoser. In this regard, a topic of great interest in this decade is how diagnosis systems perform in the presence of either sensor or communication channel fault(s). Such a problem has been named as robust diagnosis, and will be the subject of Section 3.4. In the sequel, in Section 3.5, we approach another important issue, namely how to link fault diagnosis to supervisory control so as to ensure that a system does not reach forbidden states after fault occurrence, the so-called safe diagnosis in fault tolerant control problem and its variations. Due to the growth in the use of communication networks in industrial systems, another topic of current interest is the so-called diagnosis of networked DES or simply network diagnosis. In Section 3.6 we consider the problem of fault diagnosis when a network with several channels, each one with different transmission delays, is used to transmit the signal recorded by the sensor to diagnoser. To end this chapter, we outline some open problems and present a few concluding remarks in Section 3.7.

3.2 Diagnosability and Online Diagnosis

We present, in this section, two important notions related to fault diagnosis: *(i)* diagnosability, which is a system property that ensures that all fault occurrences can be diagnosed, and; *(ii)* diagnosis, which corresponds to the actual detection of the fault occurrence. Fault diagnosis can be performed offline (when some experiment is performed and data are collected from the experiment and analyzed to verify if

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

some fault has actually occurred), or online (when the fault is detected while the system is running). We will approach, in this section, online fault diagnosis.

3.2.1 Language diagnosability

Let us consider the DES modelled by the DFA $G = (X, \Sigma, \delta, x_0)$ depicted in Fig. 3.1, where $\Sigma = \{a, b, c, d, e, \sigma_f\}$, with σ_f being the fault event whose occurrence must be detected. Notice that the language generated by automaton G is $L(G) = \overline{(ec + bca + a\sigma_f c)bd^*}$, from which we can distinguish three unbounded sequences, one that has the fault event ($s_Y = a\sigma_f cbd^n$) and two that do not possess event σ_f ($s'_N = bcabd^n$ and $s''_N = ecbd^n$), $n \in \mathbb{N}$. Let us consider the following choices for the sets of

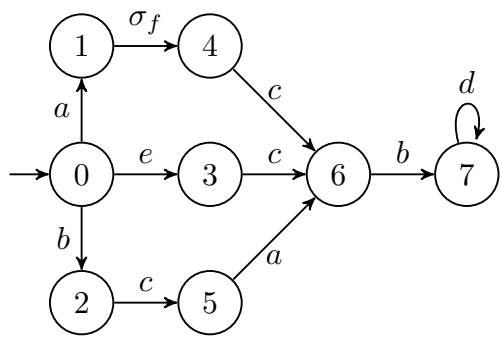


Figure 3.1: Automaton for the illustration of the concept of fault diagnosis.

observable and unobservable events, Σ_o and Σ_{uo}, respectively:

(i) $\Sigma_o = \{c, d, e\}$ and $\Sigma_{uo} = \{a, b, \sigma_f\}$. In this case, it is not possible to distinguish between the occurrence of sequences s_Y, which contains the fault event σ_f, and s'_N, which does not contain σ_f, since $P_o(s_Y) = P_o(s'_N) = cd^n$.

(ii) $\Sigma_o = \{b, d\}$ and $\Sigma_{uo} = \{a, c, e, \sigma_f\}$. In this case, the occurrence of sequences s_Y and s''_N cannot be distinguished, since $P_o(s_Y) = P_o(s''_N) = bd^n$.

(iii) $\Sigma_o = \{b, d, e\}$ and $\Sigma_{uo} = \{a, c, \sigma_f\}$. In this case, the occurrence of fault event σ_f can be detected since there does not exist any sequence s_N that does not contain the fault event and has the same projection as s_Y,

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

i.e., there does not exist s_N ($\sigma_f \notin s_N$) such that $P_o(s_N) = P_o(s_Y) = bd^n$ since $P_o(s'_N) = bbd^n$ and $P_o(s''_N) = ebd^n$.

(iv) $\Sigma_o = \{a, c\}$ and $\Sigma_{uo} = \{b, d, e, \sigma_f\}$. As in the previous case, the occurrence of fault event σ_f can always be detected. The difference here is that the projections $P_o(s_Y) = ac$, $P_o(s'_N) = ca$ and $P_o(s''_N) = c$ are bounded as opposed to cases *(i)–(iii)*, in which they are unbounded.

The above discussion shows that the occurrence of fault events cannot be detected when the chosen observable event set Σ_o makes some sequence s_Y, with an unbounded length, that contains the fault event, have the same projection over Σ_o^* as a sequence s_N ($\sigma_f \notin s_N$), with s_N not necessarily unbounded, i.e., $P_o(s_Y) = P_o(s_N)$. In other words, the detection of the fault event requires that $P_o(s_Y)$ should become different from $P_o(s_N)$ after a finite number of event occurrences once the fault takes place. For example, in case *(iii)*, when $s_Y = a\sigma_f cbd^n$ occurs, we will be sure that σ_f has occurred after the first occurrence of event d since $P_o(a\sigma_f cbd) = bd$ is different from the projection of every sequence that does not have the fault event; thus, we need to wait for three event occurrences after the fault event in order to be sure that σ_f has occurred. On the other hand, in case *(iv)*, as soon as event c occurs after the occurrence of σ_f, we are sure that σ_f has occurred.

We now present the formal definition of language diagnosability of DES, as introduced in Sampath *et al.* (1995a). To this end, let us partition the set of events Σ as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where $\dot{\cup}$ denotes disjoint union. The following notation/definition is also required.

Definition 3.1. Let L denote the language generated by an automaton, and let $\Sigma_f = \{\sigma_{f_1}, \sigma_{f_2}, \ldots, \sigma_{f_m}\} \in \Sigma_{uo}$ be the set formed with fault events σ_{f_i}, $i = 1, 2, \ldots, m$.

A. $L/s = \{t \in \Sigma^* | st \in L\}$: the post-language of L after s.

B. $\Psi(\Sigma_f) = \{s\sigma_f \in L | (s \in \Sigma^*) \wedge (\sigma_f \in \Sigma_f)\}$. With slight abuse of notation, given a trace s, the membership relation $\Sigma_f \in s$ is used to denote that $\overline{s} \cap \Psi(\Sigma_f) \neq \varnothing$.

C. (Faulty and normal sequence) A sequence $s \in L$ is a faulty sequence if $\Sigma_f \in s$. If $\Sigma_f \notin s$, then s is said to be a normal sequence.

D. (Ambiguous sequence) An unbounded faulty sequence $s_Y \in L$ is an ambiguous sequence with respect to projection P_o and σ_f if there exists a normal sequence $s_N \in L$ such that $P_o(s_Y) = P_o(s_N)$.

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

The following assumptions are also required:

A1. The language generated by G is live, i.e., for all $x \in X$, there exists $y \in X$ such that $\delta(x, \sigma) = y$ for some $\sigma \in \Sigma$.

A2. There is only one fault event, i.e., $\Sigma_f = \{\sigma_f\}$.

These assumptions are made for simplicity. Regarding Assumption **A1**, from the language diagnosability point of view, any non-live language can be made live by adding self-loops labeled with unobservable events to all states $x \in X$ such that $\delta(x, \sigma)$ is undefined for all $\sigma \in \Sigma$. As far as Assumption **A2** is concerned, if the number of fault events m is greater than one, it suffices to analyze language diagnosability with respect to each fault event σ_{f_i}, $i = 1, 2, \ldots, m$, separately and consider the remaining fault events, namely those events in $\Sigma_f - \{\sigma_{f_i}\}$ as ordinary unobservable events.

Language diagnosability is formally defined as follows (Sampath *et al.*, 1995a).

Definition 3.2 (Language diagnosability)**.** A prefix-closed and live language L, generated by an automaton G, is diagnosable with respect to projection P_o and $\Sigma_f = \{\sigma_f\}$ if the following holds true:

$$(\exists n \in \mathbb{N})(\forall s \in \Psi(\Sigma_f))(\forall t \in L/s)(\|t\| \geq n \Rightarrow D),$$

where the diagnosability condition D is

$$(\forall \omega \in P_o^{-1}(P_o(st)) \cap L)(\Sigma_f \in \omega),$$

with $\|.\|$ denoting the length of a trace. □

Thus, according to Definition 3.2, a language L is not diagnosable with respect to P_o and σ_f if for all $n \in \mathbb{N}$, there exist $s \in \Psi(\Sigma_f)$ and $t \in L/s)$, with $\|t\| \geq n$, and $\omega \in P_o^{-1}(P_o(st)) \cap L)$ such that $\sigma_f \notin \omega$. This implies L has an ambiguous sequence with respect to P_o and σ_f.

In light of Definition 3.2, we can conclude that language L generated by automaton G of Fig. 3.1 is not diagnosable with respect to P_o and σ_f in cases *(i)* and *(ii)*, but it is diagnosable in cases *(iii)* and *(iv)*.

Before we move to the next section, a few remarks are important.

1) Language diagnosability depends on the choice of the observable event set Σ_o. Notice, in case *(ii)*, where $\Sigma_o = \{b, d\}$, L is not diagnosable

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

with respect to P_o and σ_f. However, if one more event (event e) is added to Σ_o, we obtain the set of observable events $\Sigma_o = \{b, d, e\}$ of case *(iii)*, which makes L diagnosable with respect to P_o and σ_f. This has led to an important research topic usually referred to as the sensor selection problem for diagnosability (Debouk *et al.*, 2002), (Yoo and Lafortune, 2002b), (Jiang *et al.*, 2003).

2) L is diagnosable with respect to P_o and σ_f for the observable event sets $\Sigma_o = \{b, d, e\}$ and $\Sigma_o = \{a, c\}$ in cases *(iii)* and *(iv)*, respectively. For this reason, these sets of observable events are referred to as diagnosis bases (Basilio *et al.*, 2012), (Santoro *et al.*, 2017).

3) Finally, it is not difficult to see that in case *(iv)*, after the occurrence of event a, we are sure that the fault event will occur since $P_o(a) = a$ is different from any other normal sequence in L. In other words, even before the fault event σ_f actually occurs, it is possible to predict its occurrence without doubt. This observation has led to an important research topic known as fault predictability (Genc and Lafortune, 2009), (Kumar and Takai, 2010), (Khoumsi and Chakib, 2012), (Takai, 2015), (Liu, 2019), (Barcelos *et al.*, 2020).

3.2.2 Online Diagnosis

Assuming that the language generated by a DFA $G = (X, \Sigma, \delta, x_0)$ is diagnosable with respect to P_o and σ_f, a problem of practical interest is how to diagnose the occurrence of σ_f in real time, i.e., how to perform online diagnosis of DES.

In Sampath *et al.* (1995a), an automaton, named as diagnoser, was proposed to perform online diagnosis. The idea behind the construction of the diagnoser automaton is to label the states of G with labels Y (N) in order to indicate if the fault event has (respectively, has not) occurred, and, based on the occurrences of observable events only, estimate the current state of G together with the information regarding the fault event occurrence.

We will present here a slight variation of the diagnoser automaton proposed in Sampath *et al.* (1995a) that is based solely on automaton operations. We start by defining the label automaton $A_\ell = (X_\ell, \Sigma_f, \delta_\ell, x_{0,\ell})$, shown in Fig. 3.2, where $X_\ell = \{N, Y\}$, $x_{0,\ell} = N$, $\delta_\ell(N, \sigma_f) = Y$, and

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

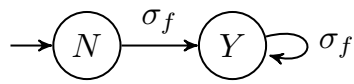


Figure 3.2: Label automaton A_ℓ.

$\delta_\ell(Y, \sigma_f) = Y$. We then compute $G_\ell = G \| A_\ell$, which has the effect of labeling all states of G prior (respectively, after) the fault occurrence with N (respectively, Y). It is worth remarking that since $\Sigma_f \subset \Sigma$, this operation is language preserving, and so $L(G_\ell) = L(G)$. It is also important to remark that some states of G can receive both labels Y and N, as long as they are in both paths, one that contains the fault event and another that does not.

The diagnoser that performs online diagnosis based only on the observation of events in Σ_o will be the observer automaton of G_ℓ with respect to Σ_o, denoted as:

$$G_d(\Sigma_o) = \text{Obs}(G_\ell, \Sigma_o) = (X_d, \Sigma_o, \delta_d, x_{0,d}), \tag{3.1}$$

where the observer states and transitions can be computed in accordance to Definition 2.2 (see also (Cassandras and Lafortune, 2009, p. 87)).

Fig. 3.3 shows automaton G_ℓ that corresponds to automaton G of Fig. 3.1, where we can see that states 6 and 7 have been split into states $6N$ and $6Y$, and $7N$ and $7Y$, respectively[1].

Figs 3.4(a) and 3.4(b) show the diagnosers associated with G assuming $\Sigma_o = \{b, d, e\}$ and $\Sigma_o = \{a, c\}$, respectively. Notice from Fig. 3.4(a) that when the diagnoser reaches state $\{7Y\}$, it is sure that fault event σ_f has occurred. On the other hand, when it reaches state $\{7N\}$, it is certain that the system is performing normally, i.e., no fault has occurred. However, after it observes the occurrence of event b, the diagnoser reaches state $\{2N, 5N, 6N, 7Y\}$. The presence of both labels Y and N in state $\{2N, 5N, 6N, 7Y\}$ indicates that the diagnoser is uncertain about the occurrence or not of the fault event σ_f.

[1]For convenience, we will replace, throughout this chapter, the notation (x, ℓ), where $x \in X$ and $\ell \in \{Y, N\}$, with simply $x\ell$; for example, we write $6N$, $7Y$, instead of $(6, N)$, $(7, Y)$, respectively.

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

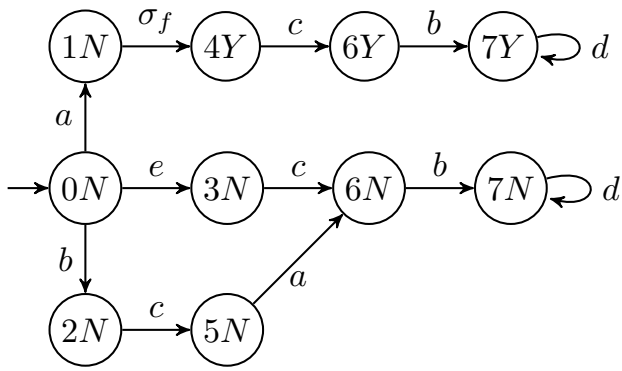


Figure 3.3: Automaton $G_\ell = G \| A_\ell$ associated with automaton G of Fig. 3.1.

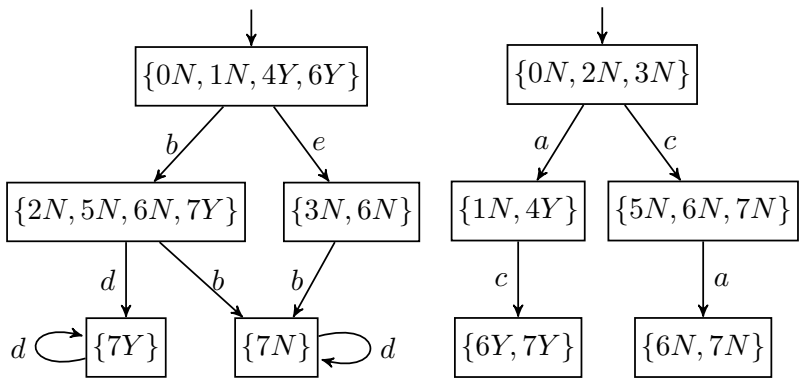
Based on the above discussion, it is possible to distinguish between the following states in the diagnoser, as far as the presence of labels Y and N are concerned (Sampath *et al.*, 1995a).

Definition 3.3 (Normal, Faulty and Uncertain States)**.** Let $x, y \in X$ and $\ell \in \{Y, N\}$. A state $x_d \in X_d$ is said to be faulty (or Y-certain), if $\ell = Y$ for all $x\ell \in x_d$, and normal (or N-certain) if $\ell = N$ for all $x\ell \in x_d$. If there exist $\{x\ell, y\tilde{\ell}\} \subseteq x_d$, x not necessarily distinct from y such that $\ell = Y$ and $\tilde{\ell} = N$, then x_d is an uncertain state of G_d.

Thus, according to Definition 3.3, state $\{7Y\}$ of diagnoser $G_{d,1}$ of Fig. 3.4(a) is a Y-certain (or faulty) state, $\{7N\}$ is an N-certain (or normal) state, and $\{2N, 5N, 6N, 7Y\}$ is an uncertain state.

It is worth remarking that both diagnosers $G_{d,1}$ and $G_{d,2}$ have succeeded in indicating if the fault has occurred or not. The only difference between them is that diagnoser $G_{d,1}$ continues observing event d after reaching states $\{7Y\}$ and $\{7N\}$ whereas diagnoser $G_{d,2}$ halts at states $\{6Y, 7Y\}$ and $\{6N, 7N\}$ since d, which is unobservable for $G_{d,2}$, is the only event that occurs in G after $G_{d,2}$ reaches those states. Such an observation allows us to conclude that although, due to Assumption **A1**, the language generated by G is live, the language generated by G_d may or may not be live. Indeed, from Eq. (3.1), $L(G_d) = P_o(L(G_\ell)) = P_o(L(G))$, and so, all suffixes formed with sequences in Σ_{uo}^* disappear when the language projection is computed.

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...



(a) Diagnoser $G_{d,1} = G_d(\Sigma_o)$, $\Sigma_o = \{b, d, e\}$.

(b) Diagnoser $G_{d,2} = G_d(\Sigma_o)$, $\Sigma_o = \{a, c\}$.

Figure 3.4: Diagnosers $G_d(\Sigma_o)$ for cases *(iii)* and *(iv)* of Section 3.2.1.

Before we move to the next section, the following remarks are worthwhile.

1) Since the diagnoser construction is based on the computation of an observer automaton, it has exponential worst case computational complexity, i.e., $O(|X_d|) = 2^{2|X|}$, where $|X|$ denotes the cardinality of the set of states of G. More recently, a diagnoser that has polynomial complexity on the size of state set of system model G has been proposed in Cabral *et al.* (2014), although not as intuitive as the diagnoser considered in this subsection. On the other hand, the diagnoser proposed in Cabral *et al.* (2014) can be easily implemented in Programmable Logic Controllers using either SFC or Ladder Languages.

2) An overview on fault diagnosis methods for discrete event systems can be found in Zaytoon and Lafortune (2013).

3.3 Diagnosability Verification

We consider in this section the problem of diagnosability verification, i.e., how to verify if the language generated by a DFA $G = (X, \Sigma, \delta, x_0)$ is diagnosable with respect to P_o and σ_f.

Diagnosability verification was first addressed in Sampath *et al.* (1995a), who proposed an automaton-based verification algorithm that,

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

besides Assumption **A1** and **A2**, also requires that no cycle of states connected by unobservable events exists in the system automaton. In addition, the diagnosability verification test presented in Sampath *et al.* (1995a) relies on the search for cycles in both the diagnoser and system automata, which may be a problem since the computational complexity to perform search for cycles is worse than exponential.

In this section, we present two tests for diagnosability verification. The first one (Viana and Basilio, 2019) builds on the diagnoser proposed in Sampath *et al.* (1995a) and requires only Assumptions **A1** and **A2** made in Section 3.2.1. We will refer to this automaton as *test diagnoser*. The second diagnosability verification test (Moreira *et al.*, 2011) employs the so-called verifier automaton and has been chosen because the idea behind its construction is intuitive and depends only on known automata operations, in addition to being the one with the least computational complexity.

Both diagnosability verification tests rely on the search for strongly connected components (SCC) in the test diagnoser and in the verifier. For this reason we recall the definition of SCC, more specifically, nontrivial SCC.

Definition 3.4 (Strongly Connected Components)**.** Given a directed graph $D = (V, E)$, where V and E denote, respectively, the set of vertices and edges of D, we say that a set of vertices $V_{scc} \subseteq V$ is a strongly connected component (SCC) of D if all pairs of vertices u and v in V_{scc} are reachable from each other, *i.e.*, $u \rightsquigarrow v$ and $v \rightsquigarrow u$, and V_{scc} is maximal, in the sense that there are no other vertices that can be included in V_{scc} that satisfy the reachability condition above. In addition, if V_{scc} is a singleton set, i.e., $V_{scc} = \{u\}$, and there does not exist any other $v \in V$, $v \neq u$, such that $u \rightsquigarrow v$ and $v \rightsquigarrow u$, then we say that V_{scc} is a nontrivial SCC if $u \rightsquigarrow u$ through a single edge (self-loop), and trivial, otherwise, (i.e., if there is no self-loop in u).

Indeed, the search of SCCs as opposed to cycles is an important feature of the methods discussed in this section since the search for cycles has factorial computational complexity (Johnson, 1975) (therefore, higher than exponential), whereas the search for SCCs is linear in the number of automaton transitions (Tarjan, 1972), (Cormen *et al.*, 2009).

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

3.3.1 Diagnosability Verification using Diagnosers

In Cassandras and Lafortune (2009), diagnosability verification is performed by constructing the diagnoser automaton in accordance with Eq. (3.1), and searching for cycles of uncertain states, i.e., states that possess both N and Y labels. If there exists such a cycle in the diagnoser, it is necessary to check if there exist two cycles of states in G_ℓ, one whose states are labeled only with Y and another one whose states are labeled only with N, both of which are associated with this particular cycle of uncertain states in the diagnoser. The language generated by the system automaton is said to be diagnosable if either the diagnoser does not have cycles of uncertain states or if it does, there do not exist both Y and N cycles in G_ℓ that can be associated with the uncertain state cycle of G_d. The following illustrates how diagnosability verification using diagnosers is performed.

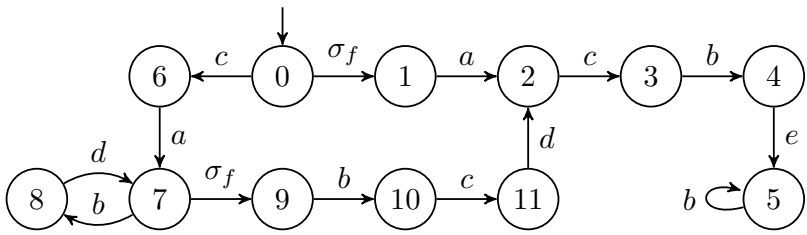
Example 3.1. Let us consider the system automata G_1 and G_2, depicted in Figs. 3.5(a) and (b), respectively, and assume we want to verify if their generated languages $L(G_1)$ and $L(G_2)$ are diagnosable with respect to P_o and Σ_f, for $\Sigma_o = \{a, b, d, e\}$ and $\Sigma_f = \{\sigma_f\}$.

Notice that both automata G_1 and G_2 have the same diagnoser automaton G_d, which is shown in Fig. 3.6. It is clear that G_d has one cycle of uncertain states, namely $\{2Y, 3Y, 7N, 9Y\}, \{4Y, 8N, 10Y, 11Y\}, \{2Y, 3Y, 7N, 9Y\}$.

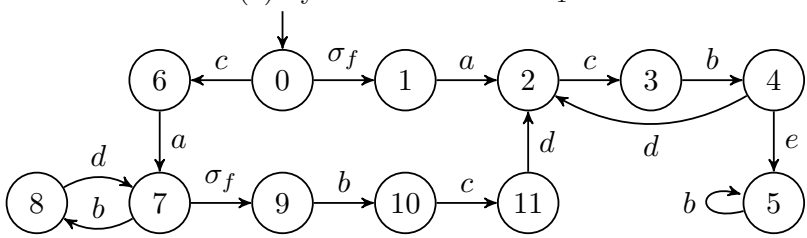
However, as we can check on Fig. 3.7(a), there are no N- and Y-labeled state cycles that can be associated with this cycle of uncertain states, and so, $L(G_1)$ is diagnosable with respect to P_o and Σ_f. On the other hand, an inspection on Fig. 3.7(b) shows that there exist two cycles of states in G_ℓ that can be associated with the cycle of uncertain states of G_d, namely, the Y- and N-labeled state cycles $\{2Y\}, \{3Y\}, \{4Y\}, \{2Y\}$ and $\{7N\}, \{8N\}, \{7N\}$, respectively. Therefore, we can conclude that $L(G_2)$ is not diagnosable with respect to P_o and Σ_f. □

More recently, the problem of diagnosability verification has been revisited in Viana and Basilio (2019), where an extension of the diagnoser proposed in Sampath *et al.* (1995a), Cassandras and Lafortune (2009), that shows in a single test automaton G_t all cycles that lead to diagnosability violation, has been proposed. The test automaton

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...



(a) System automaton G_1.



(b) System automaton G_2.

Figure 3.5: Automata G_1 and G_2 for diagnosability analysis using diagnosers.

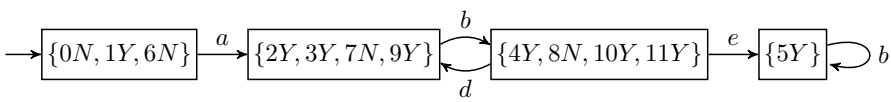


Figure 3.6: Diagnoser for system automata G_1 and G_2 of Fig. 3.5(a) and (b), assuming $\Sigma_o = \{a, b, d, e\}$.

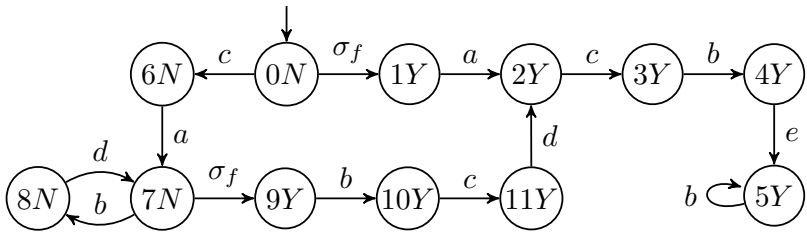
proposed in Viana and Basilio (2019) is defined as follows:

$$G_t = G_d \| G_\ell. \tag{3.2}$$

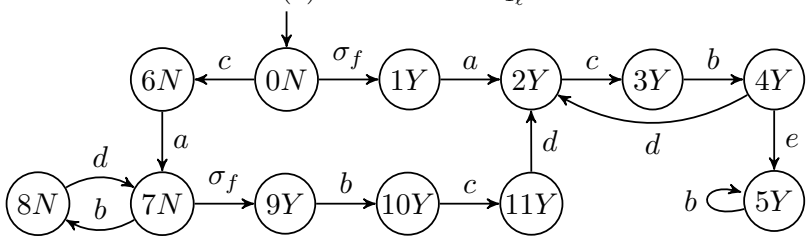
The intuition behind the construction of G_t is as follows. Since the set of events of G_d is Σ_o, which is a subset of the event set Σ of G_ℓ, $L(G_d) = P_o(L(G))$, and since $L(G_\ell) = L(G)$, it is easy to see that $L(G_t) = L(G)$, and so, the cycles of states of G_t whose second components are Y-labeled determine arbitrarily long sequences that contain the fault event. Therefore, if the first components of the states in the cycle are all formed with uncertain states of G_d, that means that the diagnoser may remain forever in doubt if the fault has occurred or not.

The diagnosability verification is performed as follows (Viana and Basilio, 2019).

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...



(a) Automaton G_{1_ℓ}.



(b) Automaton G_{2_ℓ}.

Figure 3.7: Automata G_{1_ℓ} and G_{2_ℓ} for diagnosability analysis using diagnosers.

Theorem 3.1 (Necessary and sufficient condition for language diagnosability)**.** The language L generated by automaton G is diagnosable with respect to projection P_o and Σ_f if, and only if, G_t does not have non-trivial SCCs formed with states (x_d, x_ℓ), such that x_d is uncertain and $x_\ell = xY$, where $x \in X$.

Example 3.2. Let us now illustrate the use of test automaton G_t in the verification of language diagnosability. Fig. 3.8 shows the test automaton G_{1_t}corresponding to system automaton G_1. Notice that G_{1_t} has no SCC formed with states whose first component is an uncertain state and the second has Y-labeled states, and so, $L(G_1)$ is diagnosable with respect to P_o and Σ_f, which agrees with the previous analysis carried out with diagnosers. On the other hand, from the test automaton G_{2_t} (built for system automaton G_2) shown in Fig. 3.9, we can find the SCC formed with states $(\{2Y, 3Y, 7N, 9Y\}, \{2Y\})$, $(\{2Y, 3Y, 7N, 9Y\}, \{3Y\})$ and $(\{4Y, 8N, 10Y, 11Y\}, \{4Y\})$, and since the second component of the states is a Y-labeled state, we can conclude that $L(G_2)$ is not diagnosable with respect to P_o and Σ_f, which also agrees with the previous analysis carried out using diagnosers. □

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

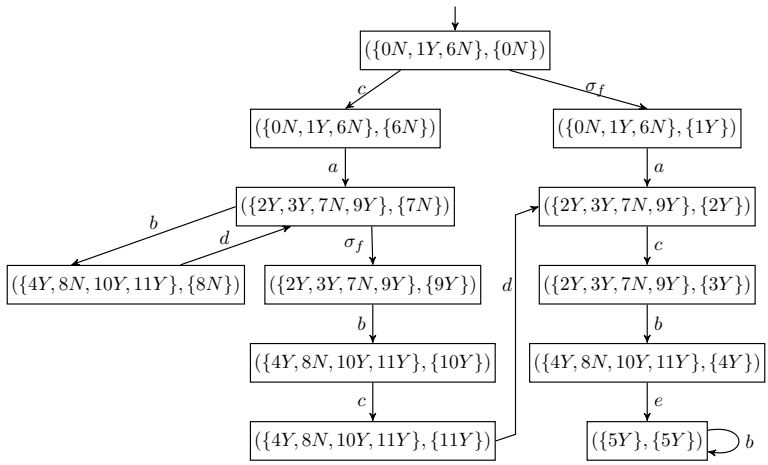


Figure 3.8: Test automaton for system automaton G_1 of Fig. 3.5(a), assuming $\Sigma_o = \{a, b, d, e\}$.

3.3.2 Diagnosability Verification Using Verifiers

Diagnosability verification performed in polynomial time was first introduced by Jiang *et al.* (2001), Yoo and Lafortune (2002a), and later on revisited in Qiu and Kumar (2006) and Moreira *et al.* (2011). The basic idea is to seek ambiguities (sequences in the language generated by the system automaton with identical projections with respect to Σ_o^*) between normal and faulty sequences. If this ambiguity persists forever for some faulty sequence, then the language is not diagnosable. On the other hand, if for all faulty sequences, the ambiguity stops after the occurrence of some observable event, then the language is diagnosable.

In this section, we will present the diagnosability verification test proposed in Moreira *et al.* (2011). The reason for choosing this approach is simply because the verifier automaton presented in Moreira *et al.* (2011) is deterministic and its construction relies solely on standard operations on automata such as co-accessibility, synchronous composition and renaming of transition labels. Interesting discussions and perspective on the verifier considered in this section can be found in Kumar and Takai (2014), Moreira *et al.* (2016).

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

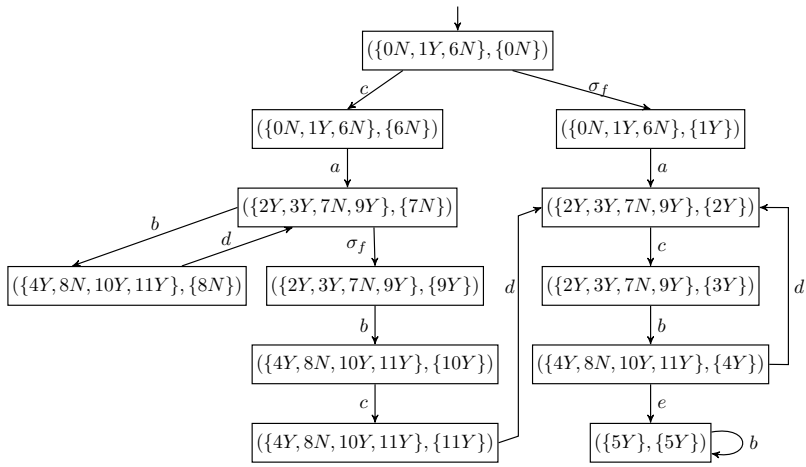


Figure 3.9: Test automaton for system automaton G_2 of Fig. 3.5(b), assuming $\Sigma_o = \{a, b, d, e\}$.

Let us start by defining automaton V as follows:

$$V = G_N \| G_Y, \tag{3.3}$$

where $G_N = (X_N, \Sigma_N, \delta_N, x_{0_N})$ and $G_Y = (X_F, \Sigma, \delta_Y, x_{0_N})$ are sub-automata of automaton G_ℓ, whose generated language is formed only with normal and faulty sequences, respectively, and can be obtained as follows: *(i)* G_N is formed by removing the transitions of G_ℓ labeled by σ_f and taking the accessible part of the resulting automaton; thus, $\Sigma_N = \Sigma - \Sigma_f$ *(ii)* G_Y is obtained by first marking all states xY of G_ℓ, then taking the coaccessible (or coreachable) part of the resulting automaton and, finally, unmarking the marked states. We will refer to automaton G_N as the system normal behaviour automaton, and to G_Y as the system faulty behaviour automaton.

Notice that automata G_N and G_Y share the same set of events, and so all of their events are synchronized, which does not allow us to distinguish between normal and faulty sequences in the search for possible ambiguities. Since ambiguity is caused by unobservable events, it is, therefore, important to find some means to differentiate normal from faulty sequences. In Moreira *et al.* (2011), such a distinction is achieved by renaming the unobservable events that appear in G_N. To

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

this end, let $\Sigma_{uo}^R = \{\sigma_R | \sigma \in \Sigma_{uo} - \Sigma_f\}$, i.e., Σ_{uo}^R is a copy of $\Sigma_{uo} - \Sigma_f$ except that the events of Σ_{uo}^R have a subscript R; for example, if $\Sigma_{uo} = \{a, b\}$, then $\Sigma_{uo}^R = \{a_R, b_R\}$. Based on that, a rename function that maps from Σ to $\Sigma_{uo} \cup \Sigma_{uo}^R$ can be defined as follows:

$$\begin{array}{rcccl} R & : & \Sigma & \rightarrow & \Sigma_o \cup \Sigma_{uo}^R \\ & & \sigma & \rightarrow & R(\sigma) = \begin{cases} \sigma, \text{ if } \sigma \in \Sigma_o \cup \Sigma_f, \\ \sigma_R, \text{ if } \sigma \in \Sigma_{uo} - \Sigma_f. \end{cases} \end{array} \tag{3.4}$$

With that in mind, the following verifier automaton is constructed:

$$G_V = G_N^R \| G_Y, \tag{3.5}$$

where $G_N^R = (X_N, \Sigma_N^R, \delta_N^R, x_{0_N})$, with $\Sigma_N^R = (\Sigma - \Sigma_{uo}) \cup \Sigma_{uo}^R$ and $\delta_N^R(x, R(\sigma)) = \delta_N(x, \sigma)$ for all events σ for which $\delta_N(x, \sigma)$ is defined. We will refer to automaton G_N^R as the system normal behaviour renamed automaton. Notice that $G_V = (X_V, \Sigma_V, \delta_V, (0N, 0N))$, where, $\Sigma_V = \Sigma \cup \Sigma_{uo}^R$ is synchronized only by the observable events, leaving the unobservable events (renamed or not) to occur freely.

Based on automaton G_V, diagnosability verification is performed as follows (Moreira *et al.*, 2011).

Theorem 3.2 (Necessary and sufficient condition for language diagnosability)**.** The language L generated by automaton G is not diagnosable with respect to projection P_o and Σ_f if, and only if, G_V has a nontrivial SCC formed with states (x_N, x_Y) where x_Y is Y-labeled for all states of the nontrivial SCC, and at least one transition connecting the states of the nontrivial SCC is labeled with an event $\sigma \in \Sigma$.

The following example illustrates both the construction of G_V and its use in language diagnosability verification.

Example 3.3. Let us use the verifier defined above to analyze the diagnosability of the languages generated by automata G_1 and G_2 depicted in Figs. 3.5(a) and (b). Notice that, as shown in Fig. 3.10, $G_{1_N}^R = G_{2_N}^R$ since the normal behaviour of system automata G_1 and G_2 are identical. Regarding the faulty behaviour automata G_{1_Y} and G_{2_Y}, it is not difficult to check that they are equal to G_{1_ℓ} and G_{2_ℓ} depicted in Figs. 3.7(a) and (b), respectively.

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

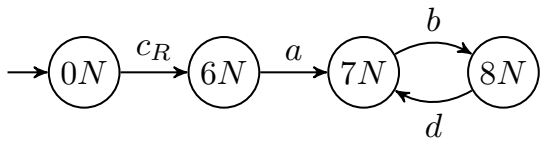


Figure 3.10: Normal behaviour renamed automata $G^R_{1_N}$ and $G^R_{2_N}$ for the system automata of Fig. 3.5(a) and (b), respectively, assuming $\Sigma_o = \{a, b, d, e\}$.

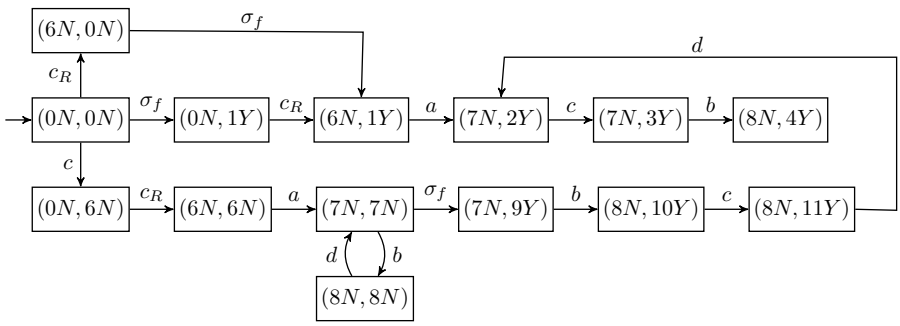


Figure 3.11: Verifier automaton G_{1_V} for the system automaton G_1 of Fig. 3.5(a), assuming $\Sigma_o = \{a, b, d, e\}$.

Fig. 3.11 shows the verifier automaton $G_{1_V} = G^R_{1_N} \| G_{1_Y}$. Notice that since G_{1_V} has no SCC formed with states whose last components are Y-labeled, we can conclude, as in the verification of language diagnosability using test automaton G_{1_t}, that $L(G_1)$ is diagnosable with respect to P_o and σ_f.

On the other hand, applying the verification test given in Theorem 3.2, we can see that $L(G_2)$ is not diagnosable with respect to P_o and σ_f. This is so because verifier G_{2_V}, depicted in Fig. 3.12, has an SCC formed with states $(7N, 2Y), (7N, 3Y), (8N, 4Y)$, whose second components are all Y-labeled states, and, in addition, all transitions connecting the states of the SCC are labeled with events in Σ. Note that only one transition labeled by an event in Σ is required according to Theorem 3.2. □

One important feature of verifier G_V is that it allows us to see how far sequence ambiguities go before they cease to exist due to the occurrence of observable events. Indeed, as we can infer from

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

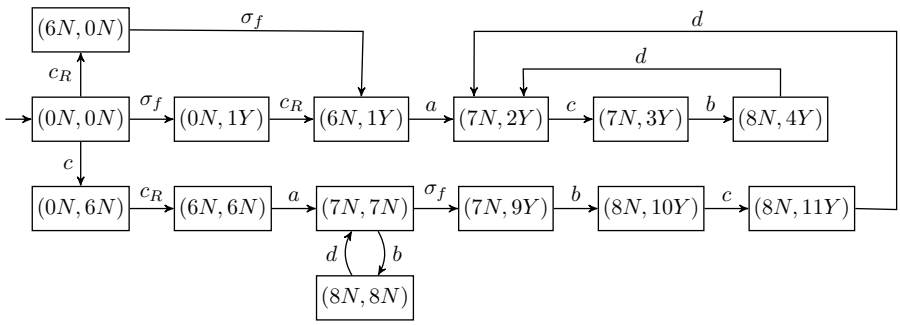


Figure 3.12: Verifier automaton G_{2_V} for the system automaton G_2 of Fig. 3.5(a), assuming $\Sigma_o = \{a, b, d, e\}$.

Fig. 3.11, $\delta_{1_V}((0N, 0N), s_V) = (8N, 4Y)$, where $s_V = \sigma_f c_R acb$, and also that $\delta_{1_V}((8N, 4Y), \sigma)$ is not defined for all $\sigma \in \Sigma_V$. Let us define the projections $P_{\Sigma_V,\Sigma} : \Sigma_V^* \to \Sigma^*$, $P_{\Sigma_V,\Sigma_N^R} : \Sigma_V^* \to (\Sigma_N^R)^*$, and denote the inverse rename function of R, defined in Eq. (3.4), as R^{-1}. As shown in Moreira *et al.* (2011), $s_N = R^{-1}(P_{\Sigma_V,\Sigma_N^R}(s_V)) = cab$ and $s_Y = P_{\Sigma_V,\Sigma}(s_V) = \sigma_f acb$ are such that $s_N, s_Y \in L(G_1)$ and also that $P_o(s_N) = P_o(s_Y) = ab$; notice that s_N takes the system automaton G_1 to state 8 whereas s_Y takes G_1 to state 4. Since the next events to occur, d or e, are both observable, the ambiguity due to traces s_Y and s_N eventually ends with the occurrence of one of these events; the occurrence of d will indicate that the fault has not occurred whereas the occurrence of event e indicates that the fault has actually occurred. This fact was used in Basilio *et al.* (2012) and Santoro *et al.* (2017) to obtain diagnosis basis[2] by searching for events that remove ambiguities associated with arbitrarily long faulty sequences.

Another important feature of verifier G_V is related to its worst case computational complexity. It is not difficult to see that, in the worst case, $|X_V| = 4|X|^2$ and $|T_V| = 8|\Sigma||X|^2$, and so, $|X_V| = O(|X|^2)$ and $|T_V| = O(|\Sigma||X|^2)$, and, thus, the complexity for the verification of diagnosability is polynomial.

[2] A diagnosis basis is a set of observable events $\Sigma_b \subseteq \Sigma$ that makes the language diagnosable with respect to P_{Σ,Σ_b} and Σ_f. In practice, an unobservable event can be made observable by adding a sensor that records its occurrence.

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

3.4 Robust Diagnosis of DES Subject to Intermittent Loss of Observations

In the previous sections, the problem of diagnosability and online diagnosis of DES is approached assuming that everything in the system works perfectly. As discussed in Section 3.2.1, language diagnosability depends on the set of observable events, say Σ_o, and when the language generated by the system automaton is diagnosable with respect to P_o, a diagnoser automaton, whose set of events is Σ_o, is built in order to follow the system evolution.

The online diagnoser works as follows: at each event occurrence in the system, the diagnoser updates its current state, and, based on the type of its current state, it is certain that the fault has (respectively, has not) occurred if it is an N-certain (respectively, Y-certain) state, or will be uncertain about the fault occurrence, if the current state is uncertain. Therefore, if due to sensor malfunction, communication channel loss (Carvalho *et al.*, 2012) or cyber-attack (Carvalho *et al.*, 2018), an event occurrence does not reach the diagnoser, it is likely to give wrong information regarding fault occurrence or even halt.

The following example illustrates how loss of event observation can make the diagnoser stop working correctly.

Example 3.4. Let us consider the system automaton shown in Fig. 3.13(a), and assume that $\Sigma = \{a, b, c, d, \sigma_f\}$ is its event set, and $\Sigma_o = \{a, b, c, d\}$ and $\Sigma_f = \{\sigma_f\}$ are the sets of observable and fault events, respectively. It is not difficult to check (by applying one of the diagnosability verification tests presented in the previous section) that $L(G)$ is diagnosable with respect to P_o and Σ_f. Therefore, according to Section 3.2.2, the diagnoser automaton G_d shown in Fig. 3.13(b) will correctly perform online diagnosis of the fault event σ_f.

Let us assume, initially, that, for some $n \in \mathbb{N}$, sequence $s_1 = c\sigma_f a d^n$ has been generated and suppose that the occurrence of event c has not been recorded due to sensor malfunction. Since event c has become unobservable, the first event occurrence to be recognized by G_d is a, which takes the diagnoser state to $\{2N\}$. When the next event of s_1 occurs, the diagnoser remains in state $\{2N\}$, where it stays as long as event d continues to occur. As a consequence, G_d is sure that the fault

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

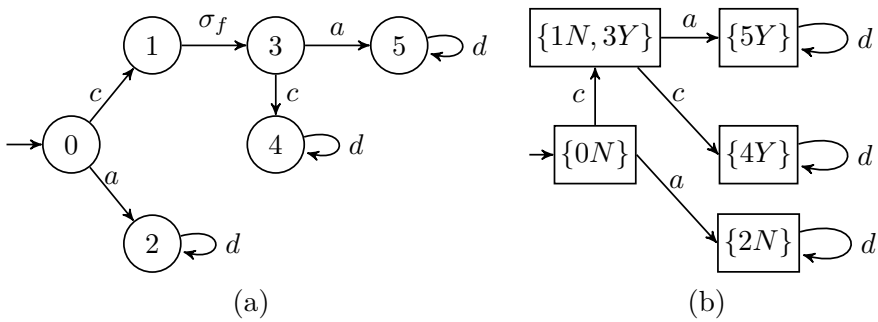


Figure 3.13: Automaton G (a) and its corresponding diagnoser G_d (b) for robust diagnosability analysis.

event σ_f has not occurred, which is wrong, since $\sigma_f \in s_1$. Assume, now, that, for some $n \in \mathbb{N}$, sequence $s_2 = c\sigma_f cd^n$ has been generated and that event c is subject to intermittent loss of observation, namely, its first occurrence is not reported to G_d but the second one is. Therefore, since the first occurrence of event c is not recognized by the diagnoser, then G_d remains in its initial state, but since the communication between the system and the diagnoser is restored before the second occurrence of event c, then when c occurs for the second time, the diagnoser moves to state $\{1N, 3Y\}$. However, the next event of s_2 to occur is d, which is not defined in $\{1N, 3Y\}$, and so the diagnoser will halt in an uncertain state. Thus, once again, it provides wrong information regarding the fault occurrence.

In this section we will consider the problem of robust diagnosability and robust diagnosis of DES subject to intermittent loss of observation. In order to address this problem, we partition the set Σ_o of observable events of system automaton G as follows:

$$\Sigma_o = \Sigma_{ilo} \dot{\cup} \Sigma_{nilo}, \tag{3.6}$$

where Σ_{ilo} is the subset of Σ_o whose events are associated with intermittent loss of observations and Σ_{nilo} is the set of events whose observation is never lost. We then, define the following set of unobservable events:

$$\Sigma'_{ilo} = \{\sigma' : \sigma \in \Sigma_{ilo}\}. \tag{3.7}$$

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

For example, for the system automaton G of Fig. 3.13(a), if we assume that $\Sigma_{ilo} = \{c\}$, then $\Sigma'_{ilo} = \{c'\}$. Notice that c' is unobservable since it models the non-observation of event c.

We will now introduce a language operation that dilates the language generated by the system automaton so as to take into account intermittent loss of observations. The idea behind this operation is to add to the original language new sequences that are identical to those that contain events in Σ_{ilo} but with the events in Σ_{ilo} replaced with the corresponding events in Σ'_{ilo}; then, since the set of events Σ'_{ilo} is unobservable, their projections are equal to ϵ, thus, effectively modeling the loss of observation of the corresponding event in Σ_{ilo}. To this end, let us define the following event set:

$$\Sigma_{dil} = \Sigma \cup \Sigma'_{ilo}, \tag{3.8}$$

where Σ'_{ilo} is built in accordance with Eq. (3.7).

Definition 3.5 (Dilation)**.** The dilation D is the mapping

$$\begin{array}{rcccl} D & : & \Sigma & \to & 2^{\Sigma_{dil}} \\ & & \sigma & \mapsto & D(\sigma) = \begin{cases} \{\sigma\}, & \sigma \in \Sigma - \Sigma_{ilo}, \\ \{\sigma, \sigma'\}, & \sigma \in \Sigma_{ilo}, \end{cases} \end{array} \tag{3.9}$$

Its extension to languages, i.e., $D : \Sigma^* \to 2^{\Sigma^*_{dil}}$, is recursively defined as follows: $D(\epsilon) = \epsilon$, and $D(s\sigma) = D(s)D(\sigma)$, for $s \in \Sigma^*$ and $\sigma \in \Sigma$.

Example 3.5. Let us consider system automaton G of Fig. 3.13(a), and assume that $\Sigma_o = \{a, c, d\}$, and that event c is subject to intermittent loss of observation; thus $\Sigma_{ilo} = \{c\}$ and $\Sigma'_{ilo} = \{c'\}$. It is not difficult to see that $L(G) = \overline{c\sigma_f(a + c)d^* + ad^*}$ is the language generated by G. Thus, according to Definition 3.5 and Eq. (3.9), $D\left(L(G)\right) = \overline{(c + c')\sigma_f(a + c + c')d^* + ad^*}$, where $c + c'$ comes from the dilation of event c, i.e., $D(c) = \{c, c'\}$.

Notice that the language that models the system behaviour in the presence of intermittent loss of observation has, besides those sequences of $L(G)$, also the following sequences: c', $c'\sigma_f$, $c'\sigma_f a$, $c'\sigma_f ad^n$, $c'\sigma_f c$, $c'\sigma_f cd^n$, $c'\sigma_f c'$, $c'\sigma_f c'd^n$, $c\sigma_f c'$ and $c\sigma_f c'd^n$, $n = 0, 1, \ldots$. It is not difficult to check that these sequences model all of the possible intermittent

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...

losses of observation of event c; for example, sequence $c'\sigma_f cd^n$ models the case when the observation of the first occurrence of c is lost but the second is not, sequence $c\sigma_f c'd^n$ models the case when the first observation is successful but the observation of the second occurrence of c is lost, and $c'\sigma_f c'd^n$ models the case when the observation of both occurrences of c are lost.

It is therefore clear that both permanent and no loss of observations are particular cases of intermittent loss of observation, since the dilation includes all sequences of $L(G)$ (no loss of observation) and permanent loss of observation (when all event observations are lost either from the start of the system operation or after some point where no previous event observation had been lost).

Let us now introduce the following projection:

$$P_{dil,o} : \Sigma^*_{dil} \to \Sigma^*_o. \tag{3.10}$$

It is not difficult to accept that, if we apply the diagnosability Definition 3.2 to $D(L(G))$ rather than to $L(G)$, and replace the usual projection P_o with $P_{dil,o}$, we will be also looking for ambiguities between faulty and normal sequences in $D(L(G))$ that did not exist in $L(G)$ since new sequences that model the intermittent loss of observation of event occurrences have been added. Consequently, it is possible that a previously diagnosable language may become non-diagnosable in the presence of intermittent loss of observations. This is the case of language $L(G)$ generated by automaton G shown in Fig. 3.13, which is diagnosable, but becomes non-diagnosable when event c is subject to intermittent loss of observation. This is so because sequences $s_N = ad^n$, $s_Y = c\sigma_f ad^n$ and $s'_Y = c'\sigma_f ad^n$ appear in $D(L(G))$ (notice that s'_Y accounts for the loss of observation of the first occurrence of c), and since $P_{dil,o}(s'_Y) = P_{dil,o}(s_N) = ad^n$, it is not possible to distinguish between the occurrence of sequences s_Y, when the observation of c is lost, and s_N. As a consequence, the language is non-diagnosable in the presence of intermittent loss of observation.

With that in mind, we can formally introduce the definition of Robust Diagnosability of DES Subject to Intermittent Loss of Observations (Carvalho *et al.*, 2012).

Documents
Document Sharing Hub
Full document details: IEEE Analysis and Control for Resilience of Discrete Event Systems: Fault Diagnosis, Op...