Identifying Behavior Models for Hybrid Production Systems

Asmir Vodenčarević

Dissertation
in Computer Science

submitted to the

Faculty of Electrical Engineering,
Computer Science and Mathematics

University of Paderborn

in partial fulfillment of the requirements for the degree of

doctor rerum naturalium
(Dr. rer. nat.)

Paderborn, March 2013

I dedicate this thesis to my family.

Acknowledgements

The thesis in hand is a product of a three-year long period of research and studies in an exciting intersection area of computer science and engineering. After summarizing the findings in this thesis, the time has come to acknowledge and thank all the people whose help and kindness I highly appreciate.

First of all, I would like to give many thanks to my supervisor, Prof. Dr. Hans Kleine Büning, for providing me with both scientific and non-scientific guidance, as well as for granting me the academic freedom to a great extent. Furthermore, I owe a great deal of gratitude to Prof. Dr. Oliver Niggemann, who played an indispensable role as my advisor and collaborator. I also thank Prof. Dr. Sandra Zilles for fruitful discussions we had and for her willingness to review this thesis.

For creating the productive and joyful working environment in the research group *Knowledge-based Systems* at the University of Paderborn, I thank my former and current colleagues: Isabela Anciutti, Simone Auinger, Maik Anderka, Gerd Brakhane, Dr. Uwe Bubeck, Dr. Markus Eberling, Dr. Thomas Kemmerich, Timo Klerx, Dr. Theodor Lettmann, Felix Mohr and Yuhan Yan. Special thanks go to Michael Baumann who read this thesis and provided many useful comments.

I am grateful to the International Graduate School *Dynamic Intelligent Systems* at the University of Paderborn, supported by the *Ministry of Innovation, Science and Research* of the federal state North Rhine-Westphalia in Germany. In particular, I would like to thank Astrid Canisius and Prof. Dr. Eckhard Steffen for their broad organizational and technical support.

I enjoyed the discussions with my colleague Alexander Maier, which resulted in various scientific publications and projects. Special thanks go to Barbara Rura and Amela Zonić for proofreading the English text and providing valuable suggestions.

Many thanks go to collaborators with whom I had inspiring discussions on different conferences and wrote a number of papers. Moreover, I express my gratitude to the anonymous reviewers who contributed to the quality of my published work.

I especially thank my friend Duško Kondor. I wish he were still with us.

Most of all, I thank my parents and my brother for their immense love and support.

Asmir Vodenčarević
Paderborn, March 2013

Abstract

The importance of safety and reliability in today's complex production systems, such as process plants, has led to the development of various anomaly detection and diagnosis techniques. Model-based approaches have established themselves among the most successful ones in the field. However, they depend on a behavior model of a system, which is typically derived manually. Manual modeling of complex production systems is a very hard task. They are characterized by both timed and probabilistic behavior. Moreover, the overall system behavior is described by discrete and continuous variables, and therefore such systems are called *hybrid systems*. Deriving behavior models for hybrid systems manually requires significant domain knowledge, budget and human resources as well as permanent manual updates. These obstacles form the so-called "modeling bottleneck".

The main goal of this thesis was to present an alternative to manual modeling, i.e. the approach to learn behavior models for hybrid systems automatically from data. Preliminary investigations have shown that for this endeavor, one should know at least the structure of the system and its discrete and continuous signals, as well as to have the measurements (logs) of those signals.

The first task included finding a suitable formalism for modeling different aforementioned characteristics of hybrid systems. To address these requirements, *hybrid automata* represent the best formalism choice. Besides their expressiveness, hybrid automata can be easily visualized, understood and interpreted by humans. To learn models for hybrid systems automatically from data, the *Hybrid Bottom-Up Timing Learning Algorithm* (HyBUTLA) was developed. The extensive complexity analysis we performed shows that the stochastic deterministic subclass of hybrid automata with one clock (for tracking time) can be learned with the HyBUTLA algorithm in time polynomial in the data size. To the best of our knowledge, the HyBUTLA algorithm is the first hybrid automata learning algorithm that models a hybrid system. We show that our approach converges to the correct automaton (the one that generated the data) when enough learning data are available.

Moreover, the *ANOmaly Detection Algorithm* (ANODA) was developed, which uses automatically learned behavior models in the anomaly detection. Our theoretical results in model-learning and anomaly detection are validated through empirical analyses performed in two real-world plants, as well as using artificial data. The obtained positive results testify to the usability of our approach in practice.

Zusammenfassung

Die Wichtigkeit von Sicherheit und Zuverlässigkeit in heutigen komplexen Produktionssystemen, wie Prozessanlagen, führte zur Entwicklung von verschiedenen Techniken zur Anomalieerkennung und Diagnose. Unter diesen haben sich modellbasierte Ansätze als am erfolgreichsten in der Branche etabliert; diese sind jedoch von einem Verhaltensmodell des Systems abhängig, das typischerweise manuell erstellt werden muss. Komplexe Produktionssysteme sind sowohl durch zeitabhängiges probabilistisches Verhalten charakterisiert. Da das Verhalten des Gesamtsystems durch diskrete und durch kontinuierliche Variablen beschrieben ist, nennt man solche Systeme *Hybrid-Systeme*. Die manuelle Modellierung solcher Systeme ist eine sehr schwierige Aufgabe: Sie erfordert umfangreiches Fachwissen, erhebliche Finanzmittel und Personal sowie ständige manuelle Updates. Diese Hindernisse bilden das sogenannte „modeling bottleneck".

Das Ziel dieser Arbeit war, eine Alternative zur manuellen Modellierung zu finden, d.h. einen Ansatz zu entwickeln, der Verhaltensmodelle von Hybrid-Systemen automatisch aus Log-Daten lernt. Voruntersuchungen haben gezeigt, dass man dafür zumindest die Struktur des Systems, seine diskreten und kontinuierlichen Signale, sowie deren Messwerte kennen sollte.

Die erste Aufgabe war, einen geeigneten Formalismus zur Modellierung der oben genannten Eigenschaften von Hybrid-Systemen zu finden. Dafür erwiesen sich *hybride Automaten* als die beste Wahl. Zusätzlich zu ihrer Ausdruckskraft haben sie den weiteren Vorteil, leicht visualisiert, verstanden und interpretiert werden zu können. Um Modelle automatisch zu lernen, wurde der *Hybrid Bottom-Up-Timing Learning Algorithmus* (HyBUTLA) entwickelt. Eine umfangreiche Komplexitätsanalyse bezüglich des Lernens von hybriden Automaten zeigt, dass ihre stochastische deterministische Unterklasse mit einem Taktgeber (zur Zeiterfassung) mit dem HyBUTLA Algorithmus in polynomieller Zeit in der Größe der Daten gelernt werden kann. Nach unserem besten Wissen ist der HyBUTLA Algorithmus der erste Lernalgorithmus für hybride Automaten, der ein Hybrid-System modellieren kann. Wir zeigen, dass unser Ansatz gegen den korrekten Automaten (derjenige, der die Daten generiert hat) konvergiert, wenn genügend Trainingsdaten verfügbar sind.

Außerdem wurde der *ANOmaly Detection Algorithmus* (ANODA) entwickelt, der erlernte Verhaltensmodelle zur Anomalieerkennung verwendet. Unsere theoretischen Ergebnisse im Lernen von Modellen sowie in der Anomalieerkennung wurden durch empirische Analysen sowohl in zwei realen Anlagen, als auch unter Verwendung von künstlichen Daten validiert. Die erhaltenen positiven Ergebnisse bezeugen die Nutzbarkeit unseres Ansatzes in der Praxis.

Contents

Chapter 1

Introduction

1.1 Motivation

On December 23, 1998 in Niihama city located in the Ehime Prefecture in Japan, a serious incident happened in the local chemical plant for production of methionine and acrolein chemical compounds [JST09]. The plant had a heat exchanger-type synthetic reactor that used the nitrate coolant for reaction control. At one point, the flow rate of the coolant started decreasing and dropped down to about a half of its nominal value. As a consequence, the temperature in the central part of the heat exchanger rose to around $500\,^{\circ}\mathrm{C}$. This caused the corrosion progression of the heat exchanger's tube that eventually punctured. Hot nitrate coolant leaked into the reactor and made contact with acrolein. Abnormal oxidation reaction occurred which irreversibly led to the explosion of the reactor, causing fire in the plant and the neighboring forest. Although no loss of life was reported, the incident caused big material and ecological damage.

This and hundreds of other industrial accidents and disasters (especially in the process industries such as chemical, petrochemical, oil, etc.) are analyzed in the Failure Knowledge Database [JST09], created by the academic community of Japan under the supervision of Japan Science and Technology Agency. In [KHH10] a statistical analysis of accident causes is performed on 364 cases in the chemical process industry, using the Failure Knowledge Database. It has been established that 73% of accidents occur due to technical failures, which include both design errors of work units and failures in components of the plant. Around 23% of accidents happen due to poor human/managerial/organizational performance and 4% of them due to unknown cause.

Having such a high percentage of technical failures in the process industry, emphasizes the need for having reliable *Fault Detection and Diagnosis* (FDD) approaches. One of the most popular and successful FDD approaches today is the *Model-Based Diagnosis* (MBD) [dKW87, Rei87]. MBD aims at detecting and localizing anomalies in a system (i.e. its components) by comparing its behavior with the behavior of a reference model. The concept of model-based anomaly detection is illustrated in Figure 1.1, where the discrepancy between observed data of a system and predicted data of a model results in a fault hypothesis. Such a MBD approach for monitoring of coolant flow rate at Niihama chemical plant could have detected the flow drop

early and signaled the anomaly to the operator, who would then have a chance to undertake an appropriate action.

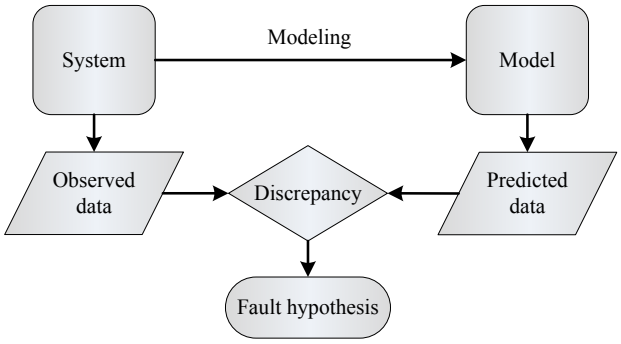


Fig. 1.1 The concept of model-based anomaly detection.

Since MBD strongly relies on a behavior model of a system, the question that naturally arises is: *Where is this model coming from?* We answer it in the following section.

1.2 The Modeling Bottleneck

This section lists common difficulties of creating behavior models of production systems. These difficulties originate in the fact that such models are today mostly created manually.

Manual modeling is a very hard task. It has to deal with the following serious problems that form the so-called "modeling bottleneck" [NMVJ11, VKBNM11b]:

Extensive expert knowledge: Sufficient knowledge about the system needs to be available. It includes the system's structure, its signals, their dependencies and mathematical relations.

Internal variables: In order to infer the laws of the system dynamics, the additional measurements of some internal variables are often required. Usually, some system internals are not observable.

Resources: Manual modeling of complex systems can be performed only by experts. Additional budget is often needed not just for their services, but also for modeling tools, software licenses, installation of sensors for required supplementary measurements etc.

Model updates: Manually created models require manual updates. In order to preserve model adequacy and accuracy, manual updates have to be carried out whenever something changes in a modeled system.

System complexity: Typical production systems, often found in the process industry, are complex dynamic systems that include a number of various hydraulic, pneumatic, electric, mechanical and chemical components, distributed architectures, embedded software, linear and nonlinear dynamics, different types of noises and external disturbances. Moreover, they are characterized by a permanent interaction between a discrete control system (which emits typically binary control

signals) and a physical process (i.e. a physical system that comprises mostly continuous process variables, such as temperature or pressure). Such systems are called *hybrid systems* [Bra05, Lyg06], and they are explained in detail in Section 2.1.

Promising alternatives to manual modeling are the data-driven approaches from the field of *computational learning theory* [HTF08]. Instead of creating behavior models of hybrid systems manually, we want to learn them automatically from data. We emphasize here that we want to learn the complete model, i.e. both the model structure (states that correspond to system modes of operation) and its parameters. This is different from the goal of the data-driven approaches coming from control engineering and the field of *system identification theory* [Lju99], which aims at identifying parameters of predefined (selected by user) mathematical relations between system variables. Our contributions are given in the following section.

1.3 Contributions of This Thesis

The goal of this thesis is to help in resolving the modeling bottleneck of hybrid production systems. The guiding research question in this endeavor is:

Given a system structure and measurements of its signals, how can behavior models for its components be identified automatically?

This question can be divided into a number of more specific questions. The most important ones are listed as follows:

1. What modeling formalism is capable of representing behavior of a typical hybrid system?
2. How can behavior models be automatically identified from data using such modeling formalism?
3. What are the complexity aspects of such an identification procedure?
4. Where and how can the available expert knowledge be utilized?
5. Are these automatically identified behavior models really usable in practice?

Answering these questions is clearly not an easy task. However, this thesis offers several answers, which represent the contributions to the fields of modeling and analysis of technical systems, computational learning theory, machine learning, and artificial intelligence in general. These contributions are:

Suitability of finite automata formalisms for learning models: A number of positive identification results exist for finite automata classes. Therefore, our research focused on learning automata models. We have found that the formalism of *hybrid automata* is capable of adequately representing hybrid production systems (Chapters 2 and 4). This formalism is expressive enough to model both state-based (discrete) and continuous dynamics of a system, as well as their interaction through the industrial networks. Additionally, it takes into account stochastic variations of production processes and timings of all changes in a system. Furthermore, hybrid automata are easily visualized, understood and interpreted by human experts.

The first hybrid automata learning algorithm: Our research resulted in an algorithm that can learn a subclass of hybrid automata, called *One-clock Stochastic Deterministic Hybrid Automaton* (1-SDHA) automatically from logged data (Chapter 6). We named our algorithm the *Hybrid Bottom-Up Timing Learning Algorithm* (HyBUTLA), and it is to date the first algorithm that can model all important characteristics of hybrid systems in the formalism of hybrid automata. It uses the same approach for learning the timing information as our *Bottom-Up Timing Learning Algorithm* (BUTLA) for learning the class of *One-clock Stochastic Deterministic Timed Automata* (1-SDTAs). The BUTLA algorithm is explained in short later on.

Thorough complexity analysis of identifying hybrid automata: Learning of automata classes is often a hard task that can be extremely computationally expensive. Since a hybrid automaton is one of the most complex types of finite automata, learning it automatically has proven to be extremely difficult. Therefore, we have conducted an extensive overview of complexity results to investigate if hybrid automata can be learned at all, and if yes, under what constrains (Chapters 3, 4 and 5). We have proven (Chapter 6) that a subclass of 1-SDHAs is identifiable by the aforementioned HyBUTLA algorithm in a reasonable time (HyBUTLA is a polynomial-time algorithm). This is possible when a sufficient amount of data is provided. Unfortunately, this amount may be overwhelming in the general case.

Expert knowledge as an asset to the learning process: The purpose of this thesis is not to replace the human experts in the tasks of modeling technical systems, but to help them in overcoming the difficulties of manual modeling. As it makes no sense to learn the already known about the system, we have defined a minimum set of expert knowledge required for our learning approach to be successful (given in the conclusions of Chapters 6 and 7). In principle, the structure of the system, its discrete and continuous signals, as well as the measurements of those signals need to be provided. Of course, when learned models are used in various application scenarios, additional application-dependent knowledge could be used further.

Use of learned behavior models in the real-world: Any research effort should justifiably be exposed to the question of its practical significance. Our research is no exception. To that aim, we have chosen to demonstrate the usability of automatically identified behavior models in the application of the model-based anomaly detection. Selection of the application is biased towards the fact that ensuring high safety and reliability standards is of the greatest importance in production systems that we model. As a part of this research, we have developed the *ANOmaly Detection Algorithm* (ANODA) that uses learned models to early detect and signal deviations of an observed system behavior from its normal operating conditions (Chapter 7). Our combined learning and anomaly detection approach has shown good performance in two real-world production systems, namely the *Lemgo Model Factory* (LMF) at the Institute Industrial IT in Lemgo, Germany, and the *Jowat AG* company in Detmold, Germany (Chapter 8). In addition, significant trials are conducted using the artificial data to experimentally validate the convergence, polynomial runtime and scalability properties of the HyBUTLA algorithm (Chapter 9).

1.4 Potential Applications

Learning behavior models of technical systems has clearly a number of potential applications, especially in model-based approaches. In this section, we list just some of them that we consider the most important ones.

Model-based diagnosis: Satisfying high requirements on safety and reliability of technical systems, which are becoming increasingly complex, represents one of the key challenges of various industries today. Therefore, a number of fault detection and diagnosis techniques have been developed over the past few decades. One of the most important ones is certainly model-based diagnosis [dKW87, Rei87] that aims at detecting anomalous behavior of a system and isolating a responsible fault. It strongly relies on a given model of a system normal behavior, which is in most cases derived manually. Our learning approach could pose a significant asset in obtaining reliable behavior models for model-based diagnosis in a more convenient, faster, and cheaper way. Due to the importance of safety aspects in production facilities, this thesis brings several important results in the application of our learning techniques in the model-based anomaly detection. These results serve to verify the applicability of our algorithms in the real-world.

Model-based design: One of the most popular approaches for designing complex technical systems is model-based design [NM10]. It is particularly suited for designing control systems and it comprises four stages, namely: (i) modeling a behavior of a running plant for which a controller is needed, (ii) designing an appropriate controller based on the obtained plant model, (iii) simulating a plant with a controller, and (iv) deploying a designed solution in the real-world. To obtain a model of a plant in the first stage of the model-based design, one can also use the data-driven approaches, i.e. our approach for learning behavior models automatically from data could represent an important asset in this application area.

Model-based testing: Testing plays a crucial role in the development of many technical systems, especially embedded systems. Model-based testing [UL07] is a technique that enables automation of the testing process through the use of a behavior model of a system under test. Having a behavior model allows generation of large pools of test cases automatically, which serve to test if certain desirable properties of a system are satisfied (e.g. dead-lock freeness in real-time systems [Ver10]). Like for model-based diagnosis and model-based design, such behavior models can also be learned automatically from data for model-based testing applications.

Optimization: Behavior models learned in the finite automata formalisms are easily visualized, understood and interpreted. This enables an easy insight into the underlying process that generated the data used for learning. Such insight could reveal previously unknown characteristics of a modeled system, which can serve for optimizing its controllers, equipment, and operating procedures. One simple example is the optimization of suboptimal energy consumption in production facilities, which is one of the crucial challenges of the European industry in the following years [Gro06]. A prediction model that simulates the normal energy consumption of a plant could actually be learned using our approach.

Applications of the field of grammatical inference: In general, the field of grammatical inference aims at finding an unknown grammar (i.e. a language, a probability distribution, or simply a model of some system) from data typically structured

in the form of strings of characters. These characters can represent a wide range of data, varying from the four bases of the *Deoxyribonucleic Acid* (DNA), over musical notes, to the letters of our human natural languages. Consequently, a broad range of grammatical inference applications emerged over time, including: discovering evolutionary interrelationships among species based on comparison of their genetic materials, natural language processing, automated translation systems for human languages, modeling styles of music composers etc. [dlH10]. Since our learning approach uses the formalism of finite automata, which are typically used in grammatical inference, it could surely find its place in such applications.

1.5 Overview

This thesis is logically divided in five parts. We give their overview as follows:

Part I Background: In the first part, we give the general background of the thesis. *Chapter 2* describes several types of technical systems and models. Special attention is devoted to known finite automata models and learning frameworks, in which they can be automatically identified from data. This chapter also provides the state of the art of popular automata learning algorithms, as well as the existing approaches to fault detection and diagnosis of hybrid production systems.

Part II Complexity of Automata Identification: This is the theoretical part of the thesis.
Chapter 3 contains definitions of several known classes of deterministic finite automata and gives an overview of the well-known negative and positive complexity results for their identification. We generalize these results to one particular class, namely *determninistic hybrid automata*, which can model certain hybrid production systems.
Chapter 4 brings definitions of known stochastic versions of aforementioned deterministic finite automata classes. It also provides several identification definitions and lists the well-known complexity results for learning stochastic deterministic automata according to these definitions. In this chapter, we also generalize the known negative identification results to the class of *stochastic deterministic hybrid automata*.
Chapter 5 introduces the reader to the existing research on polynomial approximations of stochastic deterministic automata. These approximations often serve as a workaround for the negative identification results provided in the previous chapter. In addition, several existing polynomial approximation algorithms are explained and prospects of approximating stochastic deterministic hybrid automata are discussed.

Part III Algorithms: Our main contributions are algorithms that are presented in this part.
Chapter 6 describes the existing approaches to data acquisition and processing. We explain how the data are then used for learning the class of one-clock stochastic deterministic hybrid automata. These automata can model all major characteristics of hybrid production systems. In order to learn them, we have developed the *Hybrid Bottom-Up Timing Learning Algorithm* (HyBUTLA), which we present in detail. We further analyze its properties referring to definitions and results given

in Chapter 4 and present its strengths and weaknesses. We present the proof for HyBUTLA convergence and its polynomial runtime.

Chapter 7 explains the general principle of model-based anomaly detection. We further identify and present three major types of anomalies that occur in hybrid production systems. One of our main contributions, the *ANOmaly Detection Algorithm* (ANODA) is presented in detail. We prove its real-time properties that enable it to, under certain condition, work online during the runtime of the system.

Part IV Case Studies in Learning and Anomaly Detection: This part contains comprehensive experimental results.

Chapter 8 demonstrates the usability of our algorithms in the real-world. We make a comparative empirical analysis of several automata learning algorithms using the data that come from the plant of Jowat AG company. Furthermore, we use the data from another plant, the Lemgo Model Factory located at the Institute Industrial IT in Lemgo, to evaluate our combined model-learning and anomaly detection approach based on the HyBUTLA and ANODA algorithms.

Chapter 9 brings our empirical analyses of convergence, polynomial runtime and scalability of the HyBUTLA algorithm. Fur these experiments, we are using artificially generated datasets. Convergence and runtime experiments confirm our theoretical results given in Chapter 6. Results of the scalability analysis derive several general observations, interesting for practitioners that want to apply the HyBUTLA algorithm in their fields of work.

Part V Conclusion: Chapter 10 that is given in this part, summarizes our main conclusions and contributions. Moreover, we present several possible directions for future work.

Part I

Background

Chapter 2

Foundations and State of the Art

This chapter presents foundations and the state of the art relevant for our research. First, Section 2.1 introduces the reader to the basic types of technical systems and their models. In this section, we give our definitions of system components and its parallelism model (we published them originally in [NMVJ11]). A parallelism model describes the structure of a system. Then, special attention is devoted to finite automata models and their identification frameworks in Section 2.2 and Section 2.3, respectively. Section 2.4 brings an overview of several popular algorithms for learning various stochastic finite automata (partially based on our overview given in [VMN13]). We relate to some properties of these algorithms further along this thesis. In Section 2.5, we present several existing model-based approaches to fault detection and diagnosis of hybrid production systems. Our model-learning algorithm, presented later in this thesis, could pose an important asset to these approaches, as they all depend on manually created behavior models. The chapter is finally concluded with Section 2.6, which summarizes existing gaps of aforementioned algorithms and approaches. The thesis addresses exactly these gaps.

2.1 Systems and Models

This thesis deals with identifying behavior models of technical systems automatically from data. In this section, three basic types of such systems are described, as well as models that can represent their structure and behavior.

Before the reader gets exposed to technical concepts, we first give a few general definitions of the most relevant terms in the following. We start with the term “System”.

System (according to the standard DIN 19226 of the German Institute for Standardization[1]): “...is a set of interacting or interdependent entities, real or abstract, forming an integrated whole, ... delimited from the environment with a boundary. Through the boundary the system has a connection to its surroundings...” (Material, energy and information flow).

[1] http://www.din.de/

A typical schematic representation of a system is shown in Figure 2.1. In general, every technical system comprises input signals u and output signals y.

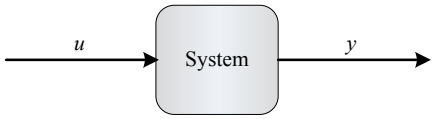


Fig. 2.1 A technical system.

The following definitions are due to online Oxford Dictionaries[2].

Model: a simplified description, especially a mathematical one, of a system or process, to assist calculations and predictions.

Behavior: the way in which a machine or natural phenomenon works or functions.

Identification: the association or linking of one thing with another.

In general, we can divide today's technical systems in three types. These are: *Discrete Event System* (DES), *continuous system*, and *hybrid system*.

Discrete event systems are those system that show a state-based behavior, i.e. they comprise a set of discrete states that represent their modes of operation and a set of events that trigger changes between those states [CL08]. In a finite time, these systems can have only a finite number of changes. They have three levels of abstraction and can correspondingly be divided in *non-timed DES* (nDES), *timed DES* (tDES), and *probabilistic (timed) DES* (ptDES) [Ver10]. nDES is completely event-driven, which means that the changes between its states can occur exclusively based on the occurrence of discrete events. In these systems, the timing of the events is not relevant. In tDES it is important at what particular time has an event happened. Timing information is associated with every event that can occur in every state of the system. ptDES is practically a tDES that has a probability function associated with every timed event, i.e. both time and probabilistic information are important. An example of DES in the real-world is a discrete control system that controls a physical plant by emitting appropriate value-discrete signals over time. Such signals are in control theory typically called *control signals*. They can be either binary (e.g. valve open/close), or have more discrete values.

Continuous systems are those systems whose belonging variables are value-continuous over time [HC01b]. Real-world examples of such variables are temperature, pressure, voltage etc. In the context of production systems, they are called *process variables*. Changes in the continuous system are typically smooth and can occur at any moment in time. In a finite time, these systems can have an infinite number of changes.

Most of today's real-world technical systems exhibit characteristics of both discrete event and continuous systems. In the process industry, they usually comprise a digital controlling mechanism embedded in a mostly analog environment [Lyg06]. These systems are called *hybrid systems* [Bra05, Lyg06]. As shown in Figure 2.2, discrete control system emits control signals to the actuators such as valves and

[2] http://oxforddictionaries.com/

drives. The change of the actuators' state (e.g. valve open/close or drive start/stop) affects process variables of the (dominantly continuous) physical system, which change continuously over time. Changes of these variables are recorded with a set of sensors that send feedback information back to the control system through the industrial network, such as PROFIBUS [Fel11]. Based on this general description, we draw out several major characteristics of hybrid production systems:

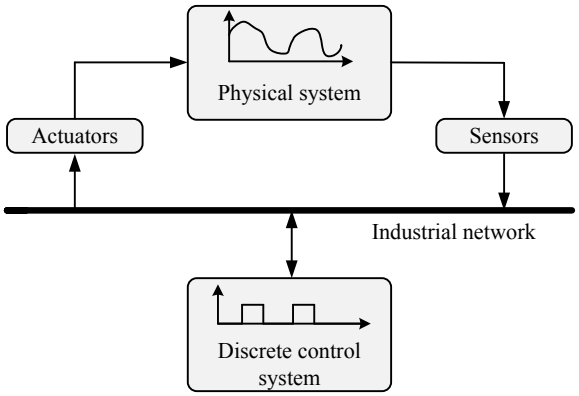


Fig. 2.2 A hybrid system.

Parallel component structure: The overall system can comprise a number of interconnected components that may work asynchronously and in parallel. These components communicate via industrial networks.

State-based (discrete) behavior: Production cycles for various products typically comprise multiple production stages. These stages are driven by the system modes of operation or its states, which are governed by a control system. A system is said to be in one of its states, when it executes one of the production stages.

Continuous signals: In addition to the state-based behavior, hybrid systems exemplify the continuous behavior as well. It is defined by changes in the process variables, i.e. signals that change continuously over time.

Interacting discrete and continuous dynamics: Changes in control signals often trigger changes in a continuous physical system through the actuators. Feedback about these changes, which is sent back to the control system via sensors, can trigger new changes in control signals.

Timed (dynamic) behavior: Since both control signals and process variables in hybrid production systems change their values over time, the behavior of such systems is time-dependent (dynamic).

Stochastic behavior: Due to many unforeseen influences, such as human factor, external disturbance, or measurement noise, production cycles of a hybrid system can significantly differ, even in the case of producing the same product. Therefore, all changes in the system should typically take place within certain confidence ranges.

Just looking at these characteristics gives a rough idea about the complexity that practitioners, who model hybrid systems, are facing.

2.1.1 Parallelism Model

The overall model of a system consists of two types of models: (i) a parallelism model of its structure, and (ii) behavior models of its components. A parallelism model defines a hierarchical set of mutually connected components that can work in parallel and asynchronously. Each single component works in a sequential manner and it does not model the parallel behavior [VKBNM11a]. Division of the system into sequential components is a prerequisite for both manual and automated derivation of behavior models, since otherwise a component model could get too large and hardly comprehensible to humans [NMVJ11]. For an illustration, consider the components A and B that work in parallel and both comprise 1000 discrete states. If A and B are modeled as a single component C, all combinations of their states are made. The resulting component C could comprise up to 1000000 states, i.e. if the parallel behavior is represent by means of a sequential behavior model, a combinational growth of model states could occur.

Parallelism models typically already exist as they are derived during plant design and planning phases. They are based on standards such as the open standard AutomationML [Aut10] for factory systems. For instance, AutomationML represents components of a plant as objects with various properties that can encapsulate other objects (components) and also be part of a larger composition. A variety of components can be represented, ranging from a simple screw, to conveyor belts and robotic arms.

In order to give a formal definition of a parallelism model, we first define system components and their types (definitions are based on our work given in [NMVJ11]).

Definition 1 (Component). A component $COMP$ is defined by a behavior function $b_C : \mathbb{R}^{m+1} \rightarrow \mathbb{R}^n, n, m \in \mathbb{N}$. b_C is a function over m input variables and over time and it returns n output variables.

Definition 2 (Discrete, Continuous, Hybrid Component). A component $COMP$ is called discrete if and only if b_C is defined as $b_C : \mathbb{R} \times \{0,1\}^m \rightarrow \{0,1\}^n, n, m \in \mathbb{N}$. I.e. b_C is defined over m boolean input variables and over time and its output variables are also boolean.

A component $COMP$ is called continuous if and only if none of its input/output variables is discrete.

A component is called hybrid if and only if it has both continuous and boolean input/output variables.

It can be noted that these definitions do not distinguish between different plant components such as *Programmable Logic Controller* (PLC), conveyor belts, etc. Such classification is interesting for domain experts, but from a learning point of view, a modeling formalism abstracts from such classifications. A parallelism model can now be formally defined as a set of connected components:

Definition 3 (Parallelism Model). A parallelism model PM is defined as a tuple $\langle \mathcal{COMPS}, pz \rangle$ where $\mathcal{COMPS} = \{COMP_0, \ldots, COMP_{r-1}\}$ is the set of components and $pz : \mathcal{COMPS} \times \mathbb{N} \rightarrow \mathcal{COMPS} \times \mathbb{N}$ maps an output variable of one component onto the input variable of another component. I.e. $pz(COMP_i, k) = (COMP_j, l)$ connects the kth output variable of $COMP_i$ with the lth input variable of $COMP_j$.

2.1.2 Behavior Model

In order to model a behavior of a system, a behavior model is required for every single component of a parallelism model. Various types of behavior models exist. In the following, we give several common classifications (due to [Kap98, HC01b]).

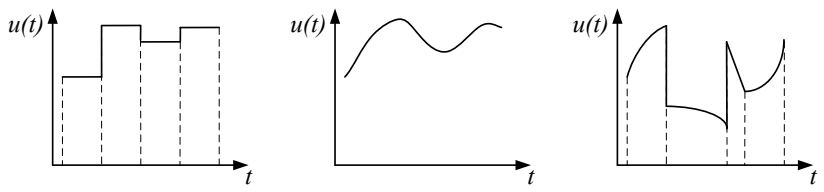
A behavior model should represent all relevant changes in a component of a system. Therefore, it has to be expressive enough to account for these changes. Based on their nature, behavior models can be classified in the following way:

Discrete models: Models that can represent non-timed, timed, and/or probabilistic discrete event systems are called discrete models. All changes take place in a finite number of discrete moments. Typical examples of such models are: Petri nets [CGS10], hidden Markov models [EAFT12], finite state automata [HZKW03], and timed automata [Tri02, SLPQ06].

Continuous models: Strict continuous systems are modeled with continuous models such as: Kalman filters [GA08, HW02a, Hen02], differential equations [Cel91], continuous Petri nets [DA87], and Bayesian networks [ZKH+05] (a nice overview is given by Narasimhan [Nar02]).

Hybrid models: Systems characterized by a constant interaction between discrete and continuous dynamics require hybrid models, which include both discrete and continuous variables. Since most of the real-world (production) technical systems are actually hybrid systems, a wide range of their modeling formalisms have been developed over years. These formalisms include: particle filters [RAG04, WD09], hybrid bond graphs [NB07], hybrid Petri nets [DA01], and hybrid automata [ACH+95, Hen96].

An example of changes in these three model types is given in Figure 2.3.



(a) Discrete model. (b) Continuous model. (c) Hybrid model.

Fig. 2.3 Example of changes in discrete, continuous, and hybrid models.

Based on the influence of stochastic variables in the system, models are divided into stochastic (probabilistic[3]) and non-stochastic:

Stochastic models: Stochastic models include the effects of randomness and their current state and output variables are described with probability functions, rather than with unique values.

Non-stochastic models: Models that do not take the effects of random variables into the account are called non-stochastic models.

[3] In this thesis the terms *stochastic* and *probabilistic* are used interchangeably.

We also distinguish between deterministic and non-deterministic models:

Deterministic models: Models whose current state depends on the previous states and provided inputs are called deterministic. Their behavior can be exactly determined, as they always provide the same outputs given the same inputs and the same set of previous states. There are no effects of randomness.

Non-deterministic models: Contrary to deterministic models, non-deterministic models could provide different outputs for the same given input and the same set of previous states.

Please note that in certain fields of machine learning, especially in grammatical inference, there is a big representation of models defined as *stochastic deterministic* (for example see the models called *stochastic deterministic finite automata* [CO94, CO99]). Such models are stochastic because they account for the stochastic processes in the system. At the same time, they are deterministic since they do not allow two or more transitions of a model from one state to different states to be triggered by the same event in the system. This thesis mostly deals with such models and they will be explained in detail in Chapter 4.

Further classification is made on static and dynamic models:

Static models: Sometimes it is enough to represent the stationary states of a system, i.e. sometimes there is no need to represent a system time evolution. For these purposes, static models are used.

Dynamic models: When the effects of time need to be modeled, a dynamic model is required. This model describes the time evolution of a system, i.e. its transient processes and changes in general. Most of the real-world production systems are modeled using dynamic models.

There are more classifications of behavior models that are not of the big importance for this thesis. These are: time-variant and time-invariant models (models whose parameters change or do not change over time), linear and nonlinear models (based on the equations that describe them), models with concentrated and distributed parameters, etc.

Based on hybrid system characteristics and model classifications given in this section, it is clear that a model we need to use, in order to learn a behavior of a hybrid system, has to be hybrid, stochastic, deterministic and dynamic.

2.2 Finite Automata Formalisms

Here we present several known finite automata models of the importance for our work in a rather informal way. As it is a generally good practice to introduce definitions once one needs them, and due to easier reading, we introduce automata classes formally in the theoretical analysis part of this thesis, namely in Chapter 3 and Chapter 4.

Significant research has been done on learning behavior models of various systems in the formalisms of finite automata. Therefore, we were interested in discovering if certain finite automata classes can (i) adequately represent the characteristics of technical systems, and (ii) if those automata classes can be automatically learned from data. In the following subsections, we present several interesting classes and relate to

characteristics of technical systems that they can model. We are interested in deterministic automata only, thus the formalism of *Non-deterministic Finite Automaton* (NFA) will not be considered[4].

2.2.1 Non-Stochastic Finite Automata

As defined earlier, non-timed discrete event system exhibits state-based behavior and its timing information is not relevant. This system can be easily modeled as the well-known *Deterministic Finite Automaton* (DFA) [HMU01]. A DFA basically consist of the states and transitions, where each transition is triggered by an occurrence of a certain event in a system. This event typically corresponds to a change in control signal of a system, such as "valve open". A DFA is deterministic in a sense that one such event can trigger only one transition from one state. A simple example of a DFA with only four states is shown in Figure 2.4. States are denoted by s_0, s_1, s_2, and s_3, while symbols a, b, and c denote the events that trigger transitions between them.

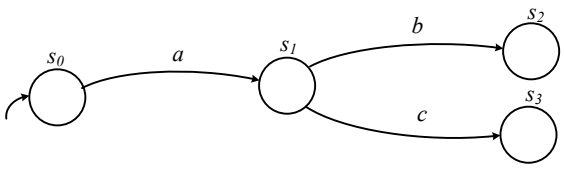


Fig. 2.4 Deterministic Finite Automaton (DFA).

Timed discrete event systems exhibit changes over time. It is clear that DFAs cannot be used to model these systems. Therefore, a formalism of *Deterministic Timed Automaton* (DTA) has been developed [AD94, VdWW08]. In addition to states and symbols, this formalism includes time intervals in which the transitions should take place. Time intervals basically model the time that a system spends in corresponding states. A DTA can have two or more transitions triggered from the same state with the same symbol, but the time intervals of these transitions cannot overlap. Therefore, these automata are deterministic. An example of a DTA is given in Figure 2.5. As before, s_0, s_1, s_2, s_3 and a, b, c, are respectively the states and the symbols that trigger transitions, while δ_1, δ_2, and δ_3 represent the transitions' time intervals.

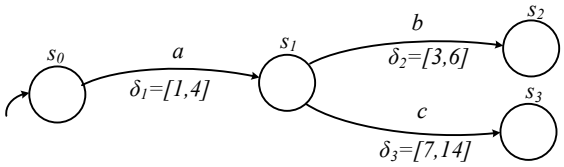


Fig. 2.5 Deterministic Timed Automaton (DTA).

[4] Significant resources on learning non-deterministic stochastic finite automata can be found at http://ai.cs.umbc.edu/icgi2012/challenge/Pautomac/.

In order to model a behavior of a hybrid system using non-stochastic deterministic finite automaton, a formalism of *Deterministic Hybrid Automaton* (DHA) has been developed [ACH+95, Hen96]. A DHA can model both control signals and continuous process variables. The latter are modeled as functions that describe a continuous evolution of a system over time. They can be in a form of differential equations [ACH+95] or various regression functions (such as for example *linear regression* [KNJ10]). Of course, DHAs are timed automata as well, since they model the timing of changes in a system. Likewise, they can have several transitions that leave a single state triggered by the same symbol, but the time intervals of those transitions cannot overlap. A simple example of a DHA is shown in Figure 2.6. In comparison to DTA shown in Figure 2.5, DHA has θ functions associated with each state. They model the behavior of the continuous variables relevant for those states.

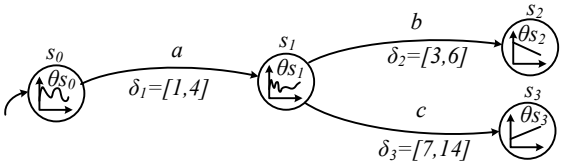


Fig. 2.6 Deterministic Hybrid Automaton (DHA).

2.2.2 Stochastic Finite Automata

Since the behavior of most technical systems is susceptible to statistical variations (e.g. due to noise, human factor, external disturbances etc.), in practice they should be modeled using stochastic models. Several stochastic finite automata exist that can respond to this task. The most simple one is the *Stochastic Deterministic Finite Automaton* (SDFA) [CO94]. A SDFA is deterministic, since it cannot include multiple transitions triggered from one state with the same symbol. In contrast to its non-stochastic relative DFA, it includes probabilities of staying in a certain state of the automaton, as well as probabilities of taking a certain transition to another state. Therefore, SDFA is a stochastic model. An example is given in Figure 2.7. When the automaton is in state s_1, it can either stay in that state with probability $p(s_1) = 0.1$, transit to state s_2 having the probability that the triggering symbol b will occur given as $p(b) = 0.2$, or transit to state s_3 with the probability of the occurrence of the triggering symbol c given as $p(c) = 0.7$. The probability $p(s_1)$ is called the *ending state probability*, while probabilities $p(b)$ and $p(c)$ represent *transition probabilities*.

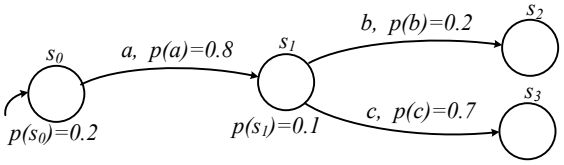


Fig. 2.7 Stochastic Deterministic Finite Automaton (SDFA).

When the timing information is added to the SDFA model, the *Stochastic Deterministic Timed Automaton* (SDTA) is obtained (an example is the *Probabilistic Deterministic Real-Time Automaton* (PDRTA) [Ver10]). This automaton is similar to DTA, but it additionally models the ending state probabilities and the transition probabilities. Figure 2.8 shows an example. It can be seen that the transitions are not only triggered by some symbol with a certain probability, but they also need to occur within certain time ranges.

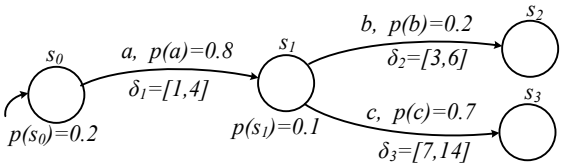


Fig. 2.8 Stochastic Deterministic Timed Automaton (SDTA).

Finally, the most complex automaton structure that we consider is the *Stochastic Deterministic Hybrid Automaton* (SDHA, see e.g. [VKBNM11a]). A SDHA (Figure 2.9) can model discrete control signals (with states and transitions), continuous signals (using θ functions), interaction between them, timing information (using the transition timing intervals), and stochastic behavior (with the transition and ending state probabilities). This class of automata is suitable for modeling the complex behavior of a hybrid production system (see Section 2.1 for an overview of hybrid system characteristics). As the example in Figure 2.9 illustrates, θ functions of SDHA can model different types of continuous dynamics, ranging from simple linear functions (θ_{s_2} and θ_{s_3}), to highly nonlinear signals (θ_{s_0} and θ_{s_1})

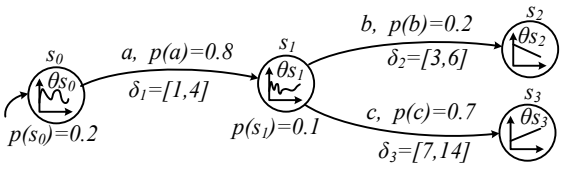


Fig. 2.9 Stochastic Deterministic Hybrid Automaton (SDHA).

To avoid confusion, in Table 2.1 we give an overview of presented non-stochastic and stochastic finite automata with respect to their ability to model certain characteristics of technical systems.

Table 2.1 Overview of presented finite automata and the features they can model.

Characteristic of a system	**Automaton**					
	DFA	DTA	DHA	SDFA	SDTA	SDHA
Discrete behavior	Yes	Yes	Yes	Yes	Yes	Yes
Continuous variables	No	No	Yes	No	No	Yes
Time	No	Yes	Yes	No	Yes	Yes
Probabilities	No	No	No	Yes	Yes	Yes

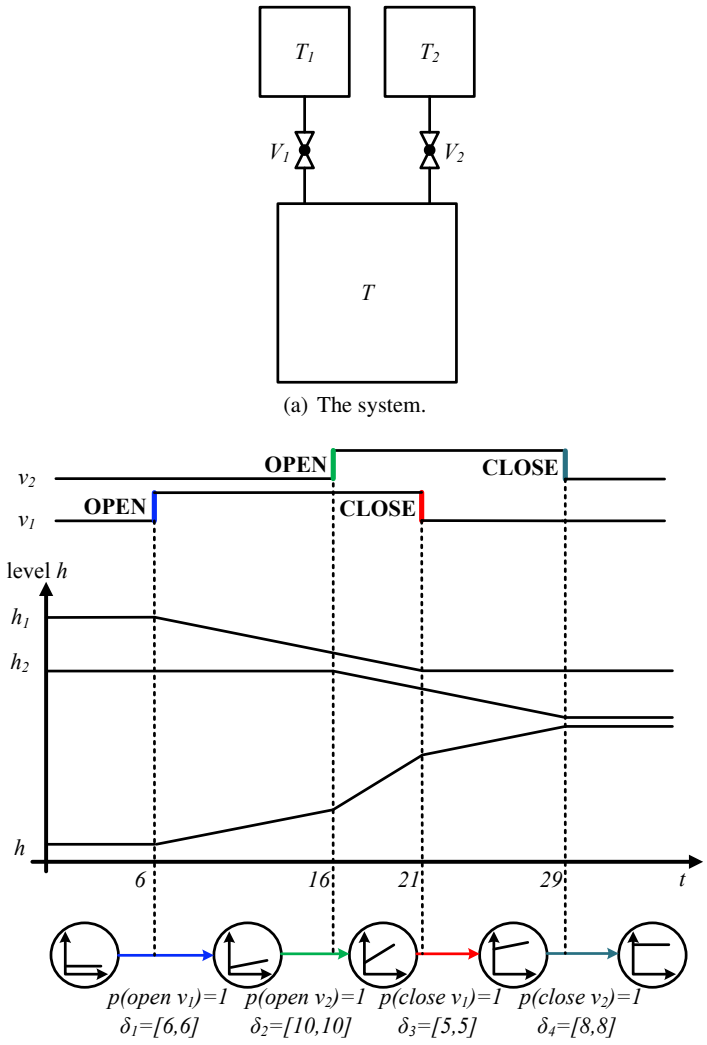


Fig. 2.10 A tank filling system, its signals, and the automaton.

Due to the importance of the SDHA formalism, we give a small practical example that illustrates its modeling power. Let us consider the tank filling system shown in Figure 2.10(a). It consists of the tanks T_1, T_2, T, on/off valves V_1, V_2, and pipes that connect them. Let us assume that tanks T_1 and T_2 contain a fluid, which can be transferred to the tank T when the corresponding valves are opened. The fluid levels in tanks T_1, T_2 and T are denoted by h_1, h_2 and h, respectively. The tank filling system is a hybrid system, as it contains discrete elements (on/off valves) that change the continuous dynamics (fluid levels in tanks). Figure 2.10(b) shows the changes in tank fluid levels h_1, h_2 and h controlled by the changes in control signals v_1 and v_2 of the valves V_1 and V_2, respectively. First, V_1 and V_2 are closed and all continuous variables (levels in three tanks) are constant. When the valve V_1 is opened, the level

h_1 decreases as the fluid is transferred to the tank T, increasing its fluid level h. At some point, the valve V_2 opens as well, which makes the increase of h significantly faster. When V_1 closes, the increase of h slows down to the previous speed (tanks T_1 and T_2 as well as valves V_1 and V_2 have the same dimensions). With the closure of V_2 the levels remain constant in all three tanks. Figure 2.10(b) additionally shows the automaton that models the behavior of this system. This automaton is derived in the following way: (i) every change in discrete control signals triggers a new automaton state, (ii) only the relative timing is used to define the transitions' timing intervals (i.e. the time is reset whenever a new state is entered), (iii) since we represent only one filling cycle, there is only one transition between every state, and therefore all transition probabilities equal to one, (iv) the automaton models the behavior of the level h in the tank T in all states. If such tank filling cycles would be repeated multiple times, this could change the transition timings and their probabilities, as two cycles would rarely be completely identical.

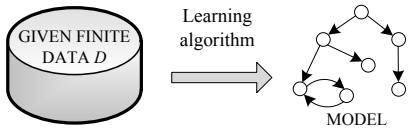
2.3 Finite Automata Identification Frameworks

In this section, we give a short overview of finite automata identification frameworks. Like in Section 2.2, our presentation is here rather informal, while the corresponding formal definitions can be found in Chapter 3, Chapter 4 and Chapter 5. Finite automata can be identified in the following frameworks:

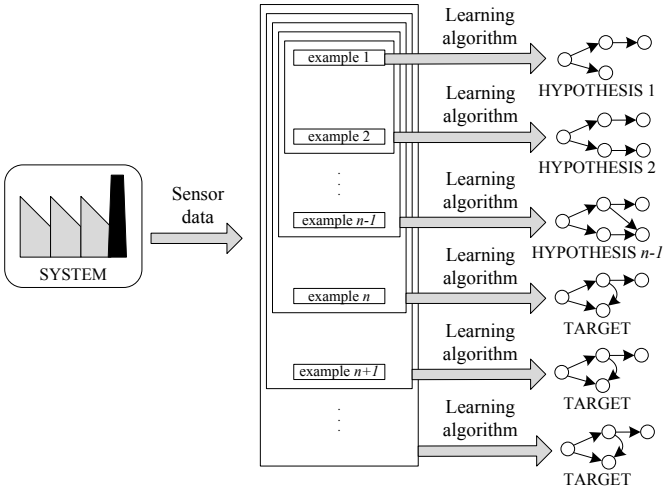
Identification from given finite data: The goal of all algorithms that learn automata from data is to find the target automaton which is consistent with these data in a sense that it correctly represents or classifies them. In modeling technical systems, the requirement is to obtain models as small and as accurate as possible. When learning data are predefined and finite, then we talk about the *identification from given finite data* [Gol78]. Acquiring additional data is in this framework not possible.

Identification in the limit: When we have the data source at the disposal, we can apply the framework of *identification in the limit* [Gol67] to learning finite automata. In theory, the data source can produce an infinite amount of data. A learning algorithm continuously receives data and correspondingly outputs different hypothesis automata. The main idea is that, provided a sufficient amount of data (also called *learning examples*), a created hypothesis automaton will eventually converge to the target automaton and will not change further when the algorithm receives more data. This identification scenario is realistic in modeling hybrid production systems, as logs from more production cycles (used as learning examples) can be recorded and used if needed. To strike out the main difference between the identification from given finite data and the identification in the limit, we have illustrated these identification frameworks in Figure 2.11(a) and Figure 2.11(b), respectively.

PAC-learning: While the goal of both previously described frameworks is to learn the exact target automaton from data, the framework of *Probably Approximately Correct learning* (PAC-learning) [Val84] actually identifies its adequate approximation. In other words, a small, predefined deviation between the identified hypothesis and the target automata is allowed with a certain probability. A deviation can be even larger than its predefined value, but only with a very small



(a) Identification from given finite data.

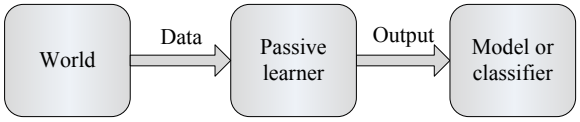


(b) Identification in the limit.

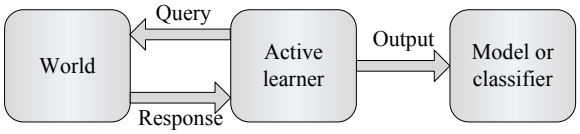
Fig. 2.11 Identification from given finite data and identification in the limit.

probability. PAC-learning represents the alternative to the learning in the limit, especially in cases (i) when it is too hard to obtain the exact target automaton, (ii) when the amount of data required for convergence is overwhelming, and (iii) when the algorithm runtime is inefficient.

Query learning: All previous three identification frameworks belong to the type of learning called *passive learning*. The learning algorithms use provided sampled data (i.e. given learning examples or logs from some data source) and output the exact target automaton or its approximation. They do not have any influence on the way the data are sampled. In contrast, *query learning* [Ang88b] represents the *active learning* [Set10] where the learning algorithm can influence or choose the data it receives. It assumes the existence of an *Oracle*, which receives and answers the queries coming from the algorithm. The Oracle typically directs a learning process by answering the algorithm's query if a certain learning example is consistent with the target automaton or not. It is generally assumed that the Oracle cannot make a mistake, and it can answer any query that the algorithm is allowed to ask. The difference between passive and active learning is illustrated in Figure 2.12 (due to Tong [Ton01]).



(a) General schema for a passive learner.



(b) General schema for an active learner.

Fig. 2.12 Difference between passive and active learning [Ton01].

2.4 Algorithms for Learning Stochastic Finite Automata

In contrast to a number of various available behavior models for technical systems (such as Bayesian and Petri networks, Bond graphs, Kalman filters etc., see Section 2.1), only a few attempts have been made towards learning them automatically from data. However, significant work is done in learning finite automata models from data (types of learning data are discussed in Subsection 2.4.2). Existing algorithms learn the structure of automata, i.e. their states and transitions. Since stochastic models are in the focus of this thesis, in Subsection 2.4.3 we give an overview of several algorithms that learn stochastic finite automata (partially based on the material we published in [VMN13]).

2.4.1 State Merging Approach to Learning

Learning stochastic finite automata was a step in the field of grammatical inference[5] that naturally followed the first algorithms for learning (non-stochastic) deterministic finite automata. Consequently, this step used the main DFA learning idea that is known as the *state merging approach to learning*. Although this approach yielded famous and successful DFA learning algorithms such as the *Regular Positive and Negative Inference* (RPNI) [OG92], it is important to note that the first such algorithms (Gold's algorithm [Gol78] and Angluin's L* [Ang87]) did not use it.

The main idea of the state merging approach to learning (and of the RPNI algorithm) is to represent the learning data in a form of a tree with states and transitions, and then to gradually create an automaton by finding and merging similar states. In this way, a generalization takes place, as an automaton gets consistent with both seen and unseen data (learning examples). The state merging approach to learning is illustrated in an industrial context in Figure 2.13, and its main steps are explained in the following.

Step 0: First, all relevant system signals are measured over multiple production cycles of a system and logged in a database. For timed systems, logs also include

[5] For a bibliographical study of grammatical inference please refer to [dlH05].

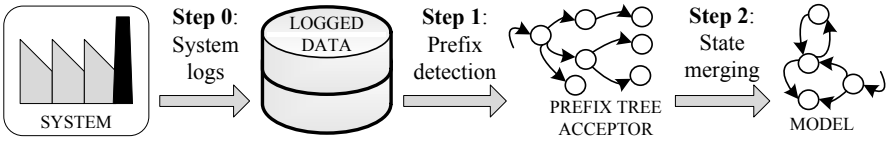


Fig. 2.13 State merging approach to learning automata.

the time stamps. For hybrid systems, logs include both measurements of control signals and continuous physical variables. The data acquisition technologies that can be used are described in [PKN+12].

Step 1: In this step, an initial automaton, typically called the *Prefix Tree Acceptor* (PTA)[6], is constructed in a form of a tree-like structure. Logs of one production cycle represent one automaton learning example. Each such example comprises multiple events of a system. We define events as the changes in discrete (often binary) control signals. These changes trigger transitions between automaton states. A PTA is created by combining common initial sequences of events of different examples (i.e. example prefixes, thus the name PTA). Each example represents a path in a PTA starting at the initial state, ending in one of the leaf states. The effect of combining example prefixes is that those examples share parts of PTA paths. A PTA is practically the largest automaton that describes all learning examples and it typically has very small or no generalization ability. Basically, it is just a smart way to store all learning examples.

Step 2: The second step caries out the true challenges of learning. It includes the search procedure that compares the PTA states finding those that are compatible (i.e. similar enough). Then the pairs of compatible states are merged until the underlying automaton is identified. Various algorithms use different means of searching for compatible states, as well as different compatibility criteria.

Additional step: It is worth noting that some algorithms include the third, additional step (see for example [VdWW10, VdWW12]). It is the state (transition) splitting, which can be seen as an operation opposite to state merging. Split creates a specialized while merge creates a generalized automaton. Motives for splitting the state are typically to make separate states for different behavioral types, e.g. to disjoin the states that are previously mistakenly recognized as compatible and merged or to make some merge possible.

2.4.2 Classification of Learning Algorithms

Here we present the general classification of automata learning algorithms based on their various characteristics.

Online vs. offline algorithms: Online algorithms can request additional data or information for learning during their runtime. Offline algorithms use only prerecorded datasets and initially given information for learning. This classification

[6] In different learning settings literature recognizes various terms, such as *frequency PTA* or *probabilistic PTA* (e.g. see [dlH10]). However, throughout this thesis we will simply use the more general term PTA.

corresponds to classification of the learning automata identification frameworks on passive and active learning, which is explained in Section 2.3. An example of the online learning algorithm is the already mentioned Angluin's L* [Ang87], while Gold's algorithm [Gol78] works in an offline manner (both learn DFAs).

Given finite vs. infinite data: Another classification of the algorithms is based on different identification frameworks that they use (see Section 2.3). Input data for learning can consist of either a given finite or an infinite number of learning examples. Gold showed that using the former setting, even the problem of learning the simplest DFA with k states from given finite data is NP-complete [Gol78]. This negative identification result generalizes to more complex automata classes. However, a number of algorithms have been developed that can converge to a target automaton when an infinite data source is present (learning in the limit [Gol67], see Section 2.3). A number of algorithms that learn stochastic finite automata learn in the limit. These algorithms are the ALERGIA [CO94] and MDI [TDdlH00] that learn SDFAs, and RTI+ [VdWW10] and BUTLA [MNV$^+$11] that learn SDTAs with one clock for tracking the time evolution (the clock resets at every transition). Their key ideas are explained in the following subsection, as they are relevant for our own approach to learning stochastic deterministic hybrid automata with one clock.

Informant vs. text identification: *Informant identification* is a type of supervised learning where both learning examples that are consistent with the target automaton (called *positive examples*, i.e. examples that are generated by the target automaton) as well as those that are not (called *negative examples*) are given to the learning algorithm. Learning from informant is probably the best-studied topic of grammatical inference, which resulted in a number of theoretical and empirical results over years. Some of them are the subject of Chapter 3. In contrast, *text identification* allows learning from positive examples only. It is considered to be one of the purest and most basic problems of grammatical inference, from which many other problems are derived [dlH10]. The motivation for learning models from text in the industrial context is the fact that the production of a faulty product happens rarely and therefore it is hard to collect sufficient amount of negative learning examples. Four aforementioned algorithms, namely ALERGIA, MDI, RTI+, and BUTLA learn from text, while algorithms RPNI (for learning DFAs) and ID_1DTA [VdWW09] (for learning DTAs with one clock) learn from informant.

Top-down vs. bottom-up merging order: In the state merging approach to learning, a learning algorithm searches for compatible states with the purpose of merging them. The order of this search has significant influence on the learning performance, especially the algorithm runtime [NSV$^+$12]. Algorithms can use either a top-down or a bottom-up merging order. In the former, compatibility of the PTA states is tested starting from the initial state, and proceeding towards the leaf states. If two states are found to be compatible, the compatibility of their belonging subtrees is recursively also checked for compatibility. This scenario is shown in Figure 2.14 (left). Subtrees t_1 and t_2 have to be compared, before the two compatible states s_1 and s_2 could be merged. In the bottom-up order, the merging process starts at leaf states and moves towards the initial state. In this way, the recursive checks for compatibility of large subtrees are minimized (see Figure 2.14 (right)). The ALERGIA is an example of an algorithm that uses the top-down merging order. The BUTLA algorithm is an example of a bottom-up merging algorithm.

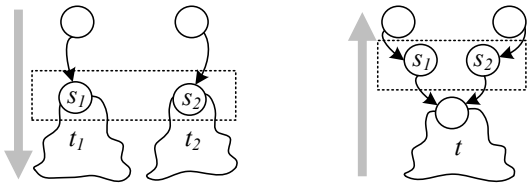


Fig. 2.14 Top-down (left) and bottom-up (right) merging orders.

2.4.3 Learning Algorithms in a Nutshell

Here we briefly describe four existing algorithms for learning stochastic finite automata from data. Their overview is given in Table 2.2, as well as the automata models of corresponding systems that they learn. In addition, we briefly present the first approach towards learning a specific subclass of hybrid automata, called *Cycle-Linear Hybrid Automata* (CLHAs).

Table 2.2 Four learning algorithms, automata models and modeled systems.

Learning algorithm	ALERGIA	MDI	RTI+	BUTLA
Automaton model	SDFA	SDFA	SDTA with one clock	SDTA with one clock
Modeled system	nDES	nDES	ptDES	ptDES

The ALERGIA algorithm: One of the most famous algorithms for learning SDFAs is the ALERGIA algorithm given by Carrasco and Oncina in 1994 [CO94]. It follows the aforementioned state merging approach to learning. First, it builds a PTA based on the available learning examples. For every single PTA state, it calculates probabilities of stopping in that state or taking a specific transition to another state. These calculations are based on the numbers of arriving, ending, and leaving examples for that state. Let s_k denote some state of the constructed PTA. Further, let the number of the examples that arrive to state s_k be g_k, the number of the examples that end in s_k be $f_k(\#)$, and the number of examples that leave s_k with some symbol a be $f_k(a)$. Then the expressions $f_k(a)/g_k$ and $f_k(\#)/g_k$ represent the probability of leaving the state s_k with the symbol a, and the ending state probability, respectively. As the learning examples are subjected to various statistical fluctuations, compatibility of two states, candidates for merging, is decided within a confidence range. A statistical test is used to check if probabilities of leaving a state or ending in a state are for two tested states similar enough. Once these probabilities are estimated based on the observations (using the Hoeffding bound [Hoe63] given later in Chapter 6 by expression (6.2)), the compatibility between any two states s_0 and s_1 is evaluated using the following criterion:

$$\left|\frac{f_0}{g_0} - \frac{f_1}{g_1}\right| > \sqrt{\frac{1}{2}\log\left(\frac{2}{\alpha}\right)}\left(\frac{1}{\sqrt{g_o}} + \frac{1}{\sqrt{g_1}}\right), f_0, g_0, f_1, g_1 \in \mathbb{N}, \qquad (2.1)$$

where g_0, g_1 are the number of examples arriving at states s_0 and s_1 respectively, f_0, f_1 are the number of examples ending/leaving those states, and $(1-\alpha)^2, \alpha \in \mathbb{R}, \alpha > 0$ is the probability of correct decision. When this inequality is true, the difference between calculated probabilities (left side of the inequality) for states s_0 and s_1 is larger than a threshold that depends on a parameter α (right side of the inequality). Therefore, the states are not compatible and will not be merged. Otherwise, the states are marked as compatible, and then their belonging subtrees are also recursively checked for compatibility using the same test. Once the compatibility is satisfied for both the states and their subtrees, the states are finally merged and probabilities are recomputed for the newly created state. Since merging can produce non-determinism, the resulting automaton is made deterministic by merging all non-deterministic transitions and states. Then, a new pair of states is checked for compatibility. The runtime of the ALERGIA algorithm is $O(n^3)$ where n is the size of the learning data. In the average case, the runtime of the algorithm version given in [CO94] is linear in n.

From the theoretical point of view, the original ALERGIA paper [CO94] did not provide the formal proof of the algorithm convergence. However, the same authors give the modified algorithm version in 1999 called *Regular Language Inference from Positive Samples* (RLIPS) [CO99]. It was shown that the probability that expression (2.1) will return the correct decision is $(1-\alpha)^2, \alpha \in \mathbb{R}, \alpha > 0$. On the other hand, de la Higuera and Thollard argue that this result holds only if each compatibility test is independent from other tests, i.e. when different tests use states attained by different examples [HT00]. This does not hold in the general case, as some previous test could have already used these examples. Therefore, the result of some previous test would have influenced the result of a new test. De la Higuera and Thollard further give the stronger proof of ALERGIA convergence in the same paper, which accounts for the cases of dependent compatibility tests. Additionally, the paper introduces a novel method for identifying transition probabilities, based on *Stern-Brocot* trees. Please note that the convergence to the target automaton is reached only when both the automaton structure (states and transitions) and probabilities are correctly identified. We strongly relate to this fact in our own learning approach, presented later in this thesis.

Based on the algorithm classification given before, the ALERGIA algorithm is an offline algorithm that identifies a target automaton in the limit from text. It uses the top-down merging order.

The MDI algorithm: The compatibility check of the ALERGIA algorithm, based on the Hoeffding bound, represents the local merging criterion. The transition and ending state probabilities are compared for the two states candidates for merging, as well as for their corresponding subtrees. However, there is no feedback information about how different are the original and a newly obtained automata (i.e. the one obtained after merging). In contrast, another SDFA learning algorithm, namely the *Minimal Divergence Inference* (MDI) algorithm [TDdlH00] takes this global information into account. It makes a trade-off between the minimal divergence of the obtained automaton from the learning examples and the minimal automaton size. Similarly to ALERGIA, MDI first creates a PTA from the given data. Then it gradually merges its states, each time measuring how big the divergence from the data is, with respect to the automaton size. When this divergence is larger than a predefined threshold after some merge, that merge would be discarded. The MDI compatibility criterion is based on the *Kullback-Leibler divergence* (K-L divergence) between two automata (which also represent probability distributions

of learning examples). K-L divergence $D(A_1||A_2)$ between automata A_1 and A_2 is calculated as follows:

$$D(A_1||A_2) = \sum_z p(z|L_{A_1}) \log \frac{p(z|L_{A_1})}{p(z|L_{A_2})}, \tag{2.2}$$

where z represents one learning example. Let A_0, A_1, and A_2 be respectively the PTA, the automaton obtained by merging PTA states, and the automaton obtained by merging A_1 states. The divergence increment between A_1 and A_2 is given as:

$$\Delta(A_1, A_2) = D(A_0||A_2) - D(A_0||A_1). \tag{2.3}$$

Let α_M be the predefined compatibility threshold, and $|A_1|$ and $|A_2|$ the sizes of automata A_1 and A_2, respectively (in the number of states). Then the MDI compatibility criterion is given by the following inequality:

$$\frac{\Delta(A_1, A_2)}{|A_1| - |A_2|} < \alpha_M. \tag{2.4}$$

When this inequality is met, the divergence increment between A_1 and A_2 is small enough relative to the size reduction, and the merge will be permanently kept (not discarded). Otherwise, the search for compatible states will continue through the automaton A_1. The time complexity of the MDI algorithm reported in [TDdlH00] is $O(n^2)$, where n is the size of the input data. Therefore, it is faster than ALERGIA. Several experimental results indicate that MDI significantly outperforms ALERGIA [dlH05, VTdlH$^+$05b]. Unfortunately, the MDI algorithm lacks the formal proof of convergence.

The MDI algorithm can be classified the same way as the ALERGIA, i.e. it is an offline algorithm that learns SDFAs in the limit from text. It uses the top-down merging order as well.

The RTI+ algorithm: The first algorithm for learning stochastic deterministic timed automata (SDTAs) is given by Verwer in 2010 [VdWW10]. It is called *Real-Time Identification from Positive Data* (RTI+) and it learns SDTAs with one clock that tracks the time between successive changes in the system[7]. As its name says, it learns from text, and moreover, it learns in the limit in an offline manner using the top-down merging order. This algorithm is basically an extension of the *Real-Time Identification* (RTI) algorithm [VdWW07], used for learning non-stochastic deterministic timed automata (DTAs) with one clock (that resets at every transition) from informant (i.e. from both positive and negative learning examples). In addition to the automaton structure, the RTI+ identifies two probability distributions, namely probabilities of the symbols (that trigger transitions), and time probabilities.

First, the algorithm builds a PTA. Then it checks if its states can be either merged or split. If no merge is possible due to the different timing constraints of the states' transitions, a split operation is performed such that merging becomes possible. The decision whether to perform a merge or a spit at some point is made based on the p-value of the likelihood-ratio test for state-merging or state-splitting. All statistical tests can be computed in polynomial time for every state. Moreover, the number of possible iterations of the algorithm is bounded by $2n^2 + n$, where

[7] Verwer denotes the subclass of SDTAs with one clock that resets at every transition as *Probabilistic Deterministic Real-Time Automata* (PDRTAs).

n denotes the size of the input data. Therefore, the RTI+ is a polynomial-time algorithm.

The BUTLA algorithm: The focus of this thesis is on our HyBUTLA algorithm that learns stochastic deterministic hybrid automata (SDHAs) with one clock. The close relative of this algorithm, capable of learning stochastic deterministic timed automata (SDTAs) with one clock, is called *Bottom-Up Timing Learning Algorithm* (BUTLA) [MNV+11]. In contrast to ALERGIA and MDI, it learns the timing information of the changes in a modeled system. Moreover, it uses the bottom-up merging order. The similarity is that it is an offline algorithm that also learns from positive examples only (text identification) and in the limit. The BUTLA compatibility criterion is also based on the Hoeffding bound, just like the compatibility criterion of the ALERGIA algorithm.
First, the BUTLA algorithm executes a preprocessing step in which for every event in a system (timed symbol) the *Probability Density Function* (PDF, probability over time) is computed. If the PDF is the sum of several Gaussian distributions, these distributions are reasonably treated as different events during learning. Although they have the same symbol, their timings are different enough, which indicates that they might have been generated by different processes in a modeled system. This situation is illustrated in Figure 2.15 (that we originally gave in [NSV+12]). It can be seen that, based on its different timings, the control event '**a**' results in different robot behaviors. When creating a PTA, the BUTLA algorithm does not combine two or more such events with different timings in a single transition. Moreover, these events are treated as different events during the merging step as well.

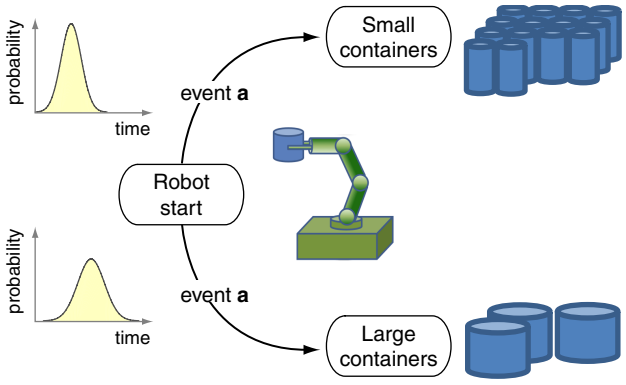


Fig. 2.15 Different robot behavior based on the different probability over time of the same event '**a**' [NSV+12].

Learning cycle-linear hybrid automata: It is worth noting that the first attempts towards learning hybrid automata were made for the *Cycle-Linear Hybrid Automaton* (CLHA) [GMY+07]. Although CLHAs do not model probabilities, they are relevant for this section as the first hybrid automata for which some learning approach was developed. Their name emphasizes their cyclic structure and modeled continuous dynamics that can be described by linear equations. CLHAs are used to model the behavior of excitable cells such as neurons in living organisms. State transitions are defined at moments when the cell electrical signal reaches its null and inflection points. Approximation of continuous dynamics is made using

the modified Prony's method [OS95] for fitting exponential functions. However, this approach cannot model discrete dynamics in the system, thus it is not suited for learning models for hybrid production systems.

2.5 Fault Detection and Diagnosis of Hybrid Systems

Some of the most important properties of hybrid production systems are reliability, safety, and efficiency. Ensuring high level of these properties is becoming increasingly hard, as hybrid systems tend to become more complex due to the employment of small, embedded components and distributed architectures. In this section, we recall several existing approaches to *Fault Detection and Diagnosis* (FDD), with an emphasis on those that are based on behavior models. Their variety exemplifies the need for obtaining such models timely and inexpensively.

2.5.1 Faults and FDD Approaches

Like any running system, a hybrid production system is also prone to faults. By fault, we consider any type of malfunction of a system component. Faults can be classified into permanent and temporary, based on the duration of their effects [Moh09]. Permanent faults cause a system to remain in a faulty state (e.g. broken pump), while temporary ones disappear after some time (e.g. provisional external disturbance). Based on a location of occurrence, faults can appear in continuous parts (e.g. pump energy overconsumption) and discrete parts (controller software error) of a system [Nar02]. Faults can also be abrupt, or subtle, causing slow system degradation [FCTB10]. A particularly difficult task is diagnosing multiple and interactive faults, especially in cases of error propagation [BS01].

To deal with some of these issues, various FDD techniques have been developed for hybrid systems. One approach, aiming at diagnosis of sensor failures, is called hardware redundancy [Ise06]. Redundant sensors are installed and fault is detected whenever a discrepancy is found in their measurements. This approach is very limited and expensive. Another model-free approach is based on expert systems [Moh09]. Domain knowledge is used to compile diagnosis rules, which enumerate symptom-fault relationships. The existence of the vast expert knowledge is presumed, which is in many cases very hard and costly to obtain. There are also other issues, like the manual update of diagnosis rules, or diagnosing complex, interacting faults [SC97].

To address the shortcomings of these approaches, *Model-Based Diagnosis* (MBD) techniques have been introduced [dKW87, Rei87]. MBD exploits the analytical redundancy [Ger98] between a model and sensor data (system). Here the model output is used as a reference to the output of a system. System faults are detected as discrepancies between these outputs. Models usually contain a reasoning mechanism for fault isolation that enables discovery of a fault source. MBD is generally applicable, easily maintainable, and has a well-founded theory [PW03]. However, applying MBD in practice faces many challenges, such as diagnosing multiple faults, diagnosing highly nonlinear (complex) systems, and dealing with noisy data. These issues are often avoided by imposing idealized assumptions [SE09].

2.5.2 Model-Based Diagnosis Using Hybrid Automata

Here we give an overview of several MBD approaches based on hybrid automata behavior models. Often, hybrid automata are in diagnosis tasks combined with other modeling formalisms.

Hybrid automata with timed Petri nets and decision trees: An approach to online diagnosis (during runtime) of hybrid systems that employs a hybrid automaton is given by Zhao et al. [ZKH+05]. A model is used for fault parameterization of both abrupt faults and subtle degradation of components. First, a model is manually created for each of the considered system components. Then, a parallel composition introduced by Alur et al. in [ACH+95] is used to construct the overall model, by combining small component models. The resulting hybrid automaton is used for generation of fault-symptom table by simulation, using predefined fault parameters (domain knowledge). The generated table is compiled into a decision tree using for example the ID3 algorithm [Qui86] in an offline manner. Online system monitoring and fault detection mechanism uses manually created timed Petri net model (see e.g. [BHR08]). Its transitions are synchronized with controller commands. Fault is detected when some autonomous event occurs outside of its expected time period. Fault identification is then evoked, which uses Petri net timing information, together with generated decision tree to direct the parameter estimation. In this way, it isolates fault candidates with high probability. Although this work presents a real-time hybrid system diagnosis, it works well only with discrete-event dynamics. Continuous system behavior is represented as a rather simple linear first-order dynamic, so a huge set of real-world nonlinear systems cannot be an object of diagnosis. It is assumed that faults happen one at the time, and that the sensor readings are not prone to errors. In addition, computation of a large fault-symptom table for all possible faults is in many real applications very hard and time-consuming process, as it requires an extensive expert knowledge. In this approach, every component had to be modeled manually before creating the overall hybrid automaton model.

A combination of hidden Markov models and Kalman filters: A general problem in diagnosing various hybrid systems is masking of fault symptoms by external disturbances or the measurement noise. One of the MBD approaches, suitable for dealing with this issue, is given by Hofbaur and Williams [HW02a]. It introduces the *Concurrent Probabilistic Hybrid Automaton* (CPHA), as a specific combination of the *Hidden Markov Model* (HMM) [Rab89] and continuous dynamic models. An overall CPHA model is a composition of CPHAs that model individual system components. It is used for hybrid estimation of a system state. In traditional approach, HMM model is used to track the changes in stochastic discrete part of a system, while the Kalman filter-based state observers [GA08] track the continuous evolution of state variables inside each of the states. In hybrid estimation, HMM transition and observation functions are modified to depend on continuous state variables, which are provided by the bank of Kalman filter estimators. This enables the estimation of the autonomous state changes, i.e. those state changes that are triggered by continuous variables, in addition to changes in control signals. A hybrid estimator is capable of tracking a number of possible system trajectories (behaviors), by employing Kalman filter for each one of them. Hybrid probabilistic functions represent fault transitions that can be simultaneously monitored. Possible computational and memory limits are addressed with a

focusing mechanism that selects only the most likely trajectories that can be taken from every current state. In this way, a significant number of relevant trajectories can be tracked in an online manner. By introducing hybrid probabilistic and hybrid observation functions, this approach can successfully deal with fault symptoms that appear on the same scale as the existing input disturbance and measurement noise. However, in large systems, a number of such trajectories could still be large. Therefore, tracking all of them would still be computationally demanding, as each trajectory requires a separate Kalman filter for estimation. In addition, some relevant fault trajectories could easily be dropped from consideration, as their probability is typically significantly smaller comparing to the probability of normal ones. On top of these remarks, the model of continuous behavior as well as the HMM need to be derived manually, which is rarely a simple task.

Hybrid automata and causal graphs: The authors extended their approach to deal with unknown system modes in [HW02b], which enabled detection of unforeseen and unknown faults. This is performed by decomposition of the overall CPHA model into interconnected partitions. First, based on the expert knowledge, a causal graph that records causal dependencies between variables of CPHA is constructed for every state. The graph is then partitioned into the independent subsystems that represent component clusters. Now unobservable and undetermined parts of the system could be excluded from the overall estimation. State variables in those parts are kept at their last known estimated values with increased variances. As decomposition is performed online, if some unobservable (or undetermined) variable becomes observable again, the estimation is restarted. This extension is very suitable for dealing with subtle component degradation, which is in general very hard to detect. However, it fails to provide any insight into the unknown behavior, as it can only detect that some unknown anomaly in one of the component clusters has happened. As for the original approach, an overall CPHA model must here also be available (i.e. manually constructed).

Hybrid automata with particle filters and a genetic algorithm: Modeling unknown fault modes of hybrid systems was also investigated by Wang and Dearden [WD09]. Here a known probabilistic hybrid automaton model of a system is presumed and unknown states are estimated using particle filters [RAG04]. If a filter is performing badly in estimating the current state, there is a high probability of a system being in a faulty state, indicating possible unknown fault. A particle filter calculates the probability of every state that is reachable from the current one. If their likelihoods are low, then most likely an unknown fault has happened. An attempt has been made to learn unknown states (i.e. fault models) and to add them to the existing model of known states. To this aim, a genetic algorithm was employed. The authors also identified equation discovery methods like *Lagramge* [TD97] to be promising in learning unknown system modes.

Hybrid automata and Expectation-Maximization algorithm: M. Henry introduces a technique for learning a hybrid model of complex physical system dynamics in [Hen02]. This has been achieved with a variant of *Expectation-Maximization* (EM) algorithm [DLR77] for parameter estimation. The algorithm comprises two steps. The first, so-called 'E' step, associates the sensor data with a set of system states (Kalman filter estimation). This task is done by the state estimation of probabilistic hybrid automaton as given in [WD09]. The second, so-called 'M' step, uses labeled data and weighted least square fit (i.e. linear regression [Wei05, MC04]) to estimate equation parameters and transition probabilities for all states in a given automaton. As this is an iterative, offline procedure, data points

are used for parameter updates until the algorithm converges. The created automaton model could then be used for online monitoring and diagnosis, as described in [WD09]. Although the algorithm performed well in the given two examples (one of them being linear and the other nonlinear system), both of them use small models, consisting of only four states. The success of this algorithm in larger systems is questionable due to two reasons. First, EM parameter estimation algorithm may easily converge to local minimum. Second, as this approach relies on a manually created automaton with given equation forms for continuous dynamics, creating states and state equations for large systems would be very hard and time consuming. Those states and equation forms are not learned automatically from data. This technique learns only equation parameters and transition probabilities.

Extended discrete event system abstraction: One of the recent approaches to diagnosis of hybrid systems that utilizes hybrid automaton model is given by Mohammadi in [Moh09]. Here a definition of hybrid automaton is extended to include states that model faulty behavior. A hybrid automaton model of a system and a bank of fault detection filters (such as Kalman filters) are firstly created separately. In the next step, they are systematically integrated to form a so-called *Extended Discrete Event System Abstraction* (EDESA). EDESA is a form of discrete event model, but it effectively includes the continuous information by using fault detection filters for observing the continuous dynamics. However, it has several drawbacks. Both hybrid automaton and fault detection filters need to be created manually. Only linear dynamics is considered. When modeling nonlinear components, linearization is performed as shown on the example of modeling a jet engine. In many real-world scenarios, linear models are simply not sufficient to describe system dynamics adequately. Finally, predefined tables of component faults (expert knowledge) are used for modeling fault states in EDESA. EDESA is capable for detecting only these faults, while adding new ones would require significant reconfiguration.

Temporal causal graphs and Kalman filters: One of the hybrid system diagnosis approaches closely related to hybrid automata is proposed by Narasimhan in [NB07]. It employs a topological, physics, and component-based modeling formalism called *Hybrid Bond Graph* (HBG) [MB98]. HBG is a combination of bond graphs, which model continuous behavior by bonds (showing energy exchanges between system components), and switched junctions that model discrete changes in the system. Junctions enable creation of both parallel and serial topologies. This modeling formalism incorporates continuous and discrete aspects of a hybrid system behavior and can be converted to hybrid automata in a systematic way [Nar02]. Other model representations can also be derived from HBG. The one used in [NB07], which shows causal and temporal variable relations is called *Temporal Causal Graph* (TCG). Monitoring of continuous behavior is realized with the extended Kalman filter as the observer. Since HBG associates one or more parameters with each modeled component, faults are detected and identified as deviations in those parameters. On the example of diagnosing an aircraft fuel transfer system, this approach gives good results in a sense of low estimation error, speed, and sensitivity to the measurement noise. The hybrid observer is capable to successfully deal with state changes in an online manner. Drawbacks are the addressing of single-time faults only and manual creation of TCGs.

By looking at these successful MBD approaches that use hybrid automata models, one common obstacle can be noticed. To a large extent, they all depend on a manually

created behavior model of a system. That is exactly the modeling bottleneck we help to overcome in this thesis. If initial hybrid automata models could be learned automatically from data, they could replace manually created models in these approaches without big efforts.

2.6 Conclusion

In this chapter, we have given an introduction to systems and models, focusing on several types of finite automata. They are interesting due to their ability to model certain types of production systems, but also because some of them can be learned automatically from data. In addition, we presented automata identification frameworks and several learning algorithms. Finally, the importance of hybrid automata is emphasized by showing various fault detection and diagnosis approaches of hybrid (production) systems, which are based on hybrid automata models.

We can conclude this chapter by identifying two major obstacles in the given state of the art, which posed a great challenge and motivation for our work and results given in the following chapters. First, the overview of current successful model-based fault detection and diagnosis approaches reveals the fact that hybrid automata (or equivalent models) are typically created manually. As argued before, manual modeling in complex production systems is a very hard task. Second, there is no algorithm that can successfully learn hybrid automata models, which are the only automaton formalism capable of representing both discrete and continuous dynamics of a hybrid system.

Part II

Complexity of Automata Identification

Chapter 3

Complexity of Identifying Deterministic Automata

In this chapter, we give an overview of the complexity analysis for the problem of identifying different classes of deterministic automata from data. Our goal is to extend and generalize the existing results specifically to the class of *Deterministic Hybrid Automata* (DHAs). Main definitions and known results presented in this chapter include:

- formal definitions of three deterministic automata classes (DFAs, DTA and DHAs) and identification settings, such as *identification in the limit*,
- existing negative results for identification of DFAs from given finite positive and negative data (Theorem 1 and Theorem 2),
- positive results for learning DFAs (Theorem 3 and Theorem 5) and DTAs with one clock (Theorem 7) in the limit from positive and negative data in polynomial time (in the size of data) and from data of polynomial size (in the size of a target automaton).

  Based on these results, we derived the following:

- we have formally generalized the aforementioned negative results to DTA and DHA classes (Proposition 1),
- we have shown that DHAs with one clock can be polynomially identified in the limit from positive and negative data (Theorem 8).

3.1 Introduction

In this section, we explain the structure of this chapter and introduce basic terms that will be formally defined in the following sections. We also present a landscape of deterministic automata identifiability from positive and negative data (informant identification), clearly marking gaps that this chapter closes in a formal way.

This chapter is organized as follows. Section 3.2 gives formal definitions of three important classes of deterministic automata: a well-known *Deterministic Finite*

Automaton (DFA), *Deterministic Timed Automaton* (DTA), and *Deterministic Hybrid Automaton* (DHA). Their relations are established by Observation 1 and Observation 2.

As mentioned before in Section 2.3, automata identification algorithms use either: (i) a finite set of given input data, or (ii) an infinite input data sequence. The former setting is analyzed in Section 3.3, which gives the well-known negative results for automaton identification and approximation problems. We have generalized them to the classes of DTAs and DHAs in Proposition 1. In the latter setting, the algorithm is given more and more data, until the convergence to some target automaton is achieved. This framework is called *identification in the limit* and formally defined in Section 3.4. Section 3.5 defines *polynomial identification in the limit* and shows the positive results for learning[1] subclasses of DTAs and DHAs with only one clock that tracks the time evolution. These classes are called *One-clock Deterministic Timed Automaton* (1-DTA) and *One-clock Deterministic Hybrid Automaton* (1-DHA). Our major positive result regarding identification of 1-DHAs is given by Theorem 8.

As explained in the previous chapter, two automata identification environments exist [Gol67], namely *informant identification* and *text identification*. Informant identification represents supervised learning, where both positive examples (belonging to the target automaton) and negative examples (not belonging to the target automaton) are presented to the learner (i.e. a learning algorithm). In text identification, only positive examples are available. In Section 3.4 it is shown that even the simplest deterministic automata–DFAs cannot be identified in the limit from text. For that reason, we restrict the analysis in this chapter to informant identification. Furthermore, the analysis is restricted here to the classes of deterministic automata. This restriction will be theoretically justified later on, by recalling the negative result for identification of non-deterministic automata (Theorem 4).

To show the scope of results given in this chapter, as well as their place in a broader picture of deterministic automata identification, Table 3.1 is given. It presents the identifiability from informant of five deterministic automata classes, with respect to two aforementioned identification settings (assuming P $\neq$ NP). Question marks denote questions that we address in this chapter.

Table 3.1 A landscape of deterministic automata identifiability from informant.

	Identification setting	
Automata	Automaton identification and approximation	Polynomial identification in the limit
DFAs	No	Yes
DTAs	?	No
1-DTAs	?	Yes
DHAs	?	?
1-DHAs	?	?

[1] The terms *learning* and *identification* are used interchangeably.

3.2 Three Classes of Deterministic Automata

In this section, we give definitions of three classes of deterministic automata, which can model certain types of technical systems. Moreover, their relations are established.

Definition 4 (Deterministic Hybrid Automaton). A determinitic hybrid automaton (DHA) is a tuple $A = (S, s_0, F, \Sigma, T, \Delta, X, \Theta)$, where

- S is a finite set of states, $s_0 \in S$ is the initial state, and $F \subseteq S$ is a set of final states.
- Σ is a finite set called the alphabet. Its elements are symbols that trigger transitions between the states.
- $T \subseteq S \times \Sigma \times S \times \Delta \times X$ is a finite set of transitions. A transition $\tau \in T$ is a tuple (s, a, s', δ, R), where $s, s' \in S$ are the source and destination states, $a \in \Sigma$ is the trigger symbol, $\delta \in \Delta$ is the timing constraint, and $R \subseteq X$ is the set of clock resets.
- $\Delta \subseteq \{\delta = [t_1, t_2] : t_1, t_2 \in \mathbb{N}\}$ is a finite set of transition timing constraints. Constraints $\delta \in \Delta$ model the time spent in a state before the transition takes place.
- X is a finite set of clocks that record the continuous time evolution. The valuation of the clock $x \in X$ is defined by $v_t(x) : X \to \mathbb{N}$.
- Θ is a finite set of functions with elements $\theta_s : \mathbb{R}^n \to \mathbb{R}^m, \forall s \in S, n, m \in \mathbb{N}$. I.e. $y = \theta_s(t, u)$ is the function computing the value changes of the output signals $y \in Y$ within state s based on the time t and values of continuous input signals $u \in U$. With Q we denote a set of discrete signals.

Figure 3.1 shows one part of deterministic hybrid automaton as an example. According to the notation from Definition 4, it follows: $s_0, s_1, s_2, s_3 \in S$ are the automaton states, $a, b, c \in \Sigma$ are the symbols whose occurrences trigger transitions, $\delta_1, \delta_2, \delta_3 \in \Delta$ are the transition timing constraints, $x_1, x_2, x_3 \in X$ are the clocks that measure time between changes in the system (i.e. occurrences of symbols) and $\theta_{s_0}, \theta_{s_1}, \theta_{s_2}, \theta_{s_3} \in \Theta$ are the functions that approximate the changes of output signals within the corresponding states. At the beginning of a production cycle, a plant (or a hybrid system in general) is in the state s_0. When the symbol a is observed, which corresponds to some change (also called *event*) in the system, the valuation $v_t(x_1)$ of the associated clock x_1 is compared with the timing constraint δ_1. Since the constraint is satisfied, the transition occurs to the state s_1. Otherwise, when the symbol b is observed and the valuation $v_t(x_2)$ of the clock x_2 satisfies the constraint δ_2, the transition occurs to the state s_2. Continuous output signals (process variables) are approximated in every state by corresponding θ functions. Multiple continuous output signals can be approximated in each state.

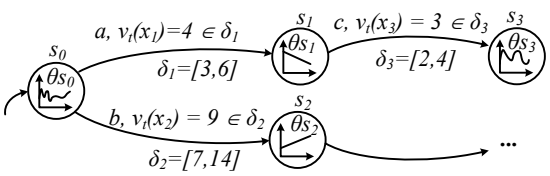


Fig. 3.1 DHA example.

Please note that based on the type of continuous dynamics that needs to be modeled, various regression methods can be used for representing θ functions, such as multiple linear regression, support vector regression or neural networks.

A deterministic timed automaton is a special kind of deterministic hybrid automaton.

Definition 5 (Deterministic Timed Automaton [VdWW08][2]). A deterministic timed automaton (DTA) is a tuple $A = (S, s_0, F, \Sigma, T, \Delta, X)$, where

- S is a finite set of states, $s_0 \in S$ is the initial state, and $F \subseteq S$ is a set of final states.
- Σ is a finite set called the alphabet. Its elements are symbols that trigger transitions between the states.
- $T \subseteq S \times \Sigma \times S \times \Delta \times X$ is a finite set of transitions. A transition $\tau \in T$ is a tuple (s, a, s', δ, R), where $s, s' \in S$ are the source and destination states, $a \in \Sigma$ is the trigger symbol, $\delta \in \Delta$ is the timing constraint, and $R \subseteq X$ is the set of clock resets.
- $\Delta \subseteq \{\delta = [t_1, t_2] : t_1, t_2 \in \mathbb{N}\}$ is a finite set of transition timing constraints. Constraints $\delta \in \Delta$ model the time spent in a state before the transition takes place.
- X is a finite set of clocks that record the continuous time evolution. The valuation of the clock $x \in X$ is defined by $v_t(x) : X \to \mathbb{N}$.

Observation 1. *A deterministic timed automaton (DTA) is a deterministic hybrid automaton (DHA) $A = (S, s_0, F, \Sigma, T, \Delta, X, \Theta)$, where $\Theta = \emptyset$.*

A well-known deterministic finite automaton is a special kind of deterministic timed automaton.

Definition 6 (Deterministic Finite Automaton [HMU01]). A deterministic finite automaton (DFA) is a tuple $A = (S, s_0, F, \Sigma, T)$, where

- S is a finite set of states, $s_0 \in S$ is the initial state, and $F \subseteq S$ is a set of final states.
- Σ is a finite set called the alphabet. Its elements are symbols that trigger transitions between the states.
- $T \subseteq S \times \Sigma \times S$ is a finite set of transitions[3]. A transition $\tau \in T$ is a tuple (s, a, s'), where $s, s' \in S$ are the source and destination states, and $a \in \Sigma$ is the trigger symbol.

Observation 2. *A deterministic finite automaton (DFA) is a deterministic timed automaton (DTA) $A = (S, s_0, F, \Sigma, T, \Delta, X)$, where $\Delta = \emptyset$ and $X = \emptyset$.*

3.3 Automaton Identification Problem

This section gives preliminaries and definitions for identifying deterministic automata from given finite positive and negative data. We cite negative results for DFAs under

[2] The original definition given in [VdWW08] differs slightly. Instead of the set of transition timing constraints Δ, transition clock guards are given.

[3] Often in the literature a set of transitions is given by the transition function $\tau : S \times \Sigma \to S$.

these identification definitions and generalize them to DTAs and DHAs in Proposition 1.

Although the field of grammatical inference in general uses the context of learning an unknown grammar from the set of learning examples [dlH10] (i.e. the language that generated those examples), the definitions and results are here given in the context of automata identification (the terms *learning* and *identification* are in this thesis used interchangeably). In the context of identification of deterministic hybrid automata, we try to answer the following general question: *Assuming an infinite or finite set of logs from some system (measurements from a plant), and an algorithm capable of approximating system continuous dynamics up to a predefined error margin, can a deterministic hybrid automaton behavior model be automatically identified?* To identify a deterministic hybrid automaton, means to identify its elements from Definition 4. Before defining this problem formally, we first explain some preliminaries and notations.

In this chapter, we are concerned with the informant identification [Gol67], where learning examples (the data) come in a form of non-conflicting sets of positive and negative examples ($\mathcal{D}_+$, $\mathcal{D}_-$), i.e. $\mathcal{D}_+ \cap \mathcal{D}_- = \emptyset$ (the learning example is defined and explained in Definition 9 and Example 1). The identified deterministic hybrid automaton model A needs to be consistent with the examples, i.e. it needs to *accept* the positive and *reject* the negative ones. The automaton is said to accept the example if the example ends in one of the automaton's final states (states from the set F, see Definition 4). Otherwise, the automaton rejects the example. The set of all examples that automaton A accepts constitutes an automaton language L_A, thus $\mathcal{D}_+ \subseteq L_A$. For a language L it holds $L \subseteq \Sigma^*$, where Σ^* is the set of all finite strings of symbols from Σ (i.e. learning examples). The set of all examples that automaton A rejects is $\overline{L} = \{z \in \Sigma^* : z \notin L\}$, thus $\mathcal{D}_- \subseteq \overline{L}$. The length of a string $z \in \Sigma^*$ is denoted by $|z|$. The unique string of zero length is denoted by λ and called an *empty string*. If for two automata A and A' from some class C it holds that $L_A = L_{A'}$, these automata are said to be equivalent (they represent the same language L).

We start the analysis of identifying deterministic hybrid automaton from given finite data by citing the major result for DFAs given by Gold in 1978 [Gol78]. We formalized the Gold's problem of *Minimum automaton identification from given data* in the following definition [4]:

Definition 7 (Automaton Identification Problem). Assuming given finite sets of positive $\mathcal{D}_+$ and negative $\mathcal{D}_-$ examples (i.e. strings over a finite alphabet) and a positive integer k (representing the minimum number of states), the automaton identification problem is the problem of finding the automaton $A \in C$ with at most k states (called the smallest automaton) from some class of finite automata C. Automaton A needs to be consistent with the data, i.e. it needs to accept all examples from $\mathcal{D}_+$ and reject all examples from $\mathcal{D}_-$.

The following theorem states Gold's result for the automaton identification problem of deterministic finite automata.

Theorem 1 ([Gol78]). *The automaton identification problem for DFAs is NP-complete.*

Proof (Sketch). It is easy to show that this problem is in NP. If a solution automaton A is given, the consistency is checked by ensuring that it accepts all examples from

[4] For more general definition of the identification problem, please see [Lai88].

$\mathcal{D}_+$ and rejects all examples from $\mathcal{D}_-$. Ensuring that the automaton has k states is done simply by counting the states. Both of these tasks take polynomial time. Showing that this problem is NP-hard is more complicated. A DFA is completely identified if its transitions are identified. In order to do so, Gold formulated the following question: *Given data with labeled positive and negative examples ($\mathcal{D}_+$, $\mathcal{D}_-$), is there a consistent finite automaton with states reachable by these examples?* This question is called *the Transition Assignment Problem* P_{TrAss}. To answer it, given data are rearranged in a form of a *state characterization matrix*. Transitions are identified by finding the missing elements of the matrix. In the proof, the missing elements are replaced by functions of literals that belong to conjunctive normal form (CNF) expressions. In this way, the satisfiability problem (SAT) with CNF expressions is reduced to the P_{TrAss} problem. Since SAT is NP-complete, it follows that the P_{TrAss} problem is NP-hard. Therefore, the automaton identification problem is both in the set of NP and NP-hard problems, i.e. it is NP-complete. □

This result raised the following interesting question: *Is there a polynomial-time algorithm that identifies approximately small DFA?* Approximately small automaton is defined in the following:

Definition 8 (Approximately Small Automaton). Assume that automaton $A \in C$ of size $|A|$ (the number of states) from some class of automata C is the smallest automaton consistent with the given finite sets of positive $\mathcal{D}_+$ and negative $\mathcal{D}_-$ examples. Let a polynomial $PolyP$ be given. An approximately small automaton $A' \in C$ is any automaton consistent with ($\mathcal{D}_+$, $\mathcal{D}_-$) for which it holds $L_{A'} = L_A$ and $|A'| \leq PolyP(|A|)$.

Pitt and Warmuth have studied the previous question with care and proved the following theorem:

Theorem 2 ([PW93]). *The problem of identifying approximately small DFA is NP-complete.*

Proof (Sketch). If such approximately small DFA were given, it would be easy to verify in polynomial time that it is consistent with given data ($\mathcal{D}_+$, $\mathcal{D}_-$). Checking if its size is at most polynomially larger than the size of the smallest consistent DFA is trivial, thus the problem of identifying approximately small DFA is in NP. The proof that this problem is NP-hard uses a similar idea as the proof of Theorem 1. It exploits the polynomial-time reduction from the 1-in-3SAT problem, to the problem of finding approximately small consistent DFA. In 1-in-3SAT problem, the input is a set of clauses, each consisting of three literals where each literal can be either a variable or its negation. In this problem, the goal is to determine if a variable truth assignment exists, such that each clause has exactly one true literal. 1-in-3SAT problem is a variant of the 3SAT problem. Both of them are known to be NP-complete [Sch78], and thus the problem of identifying approximately small DFA is also NP-complete. □

In order to show how these negative results for DFA generalize to the classes of DTAs and DHAs, we give the following proposition with a very simple proof.

Proposition 1. *The automaton identification problem for deterministic timed automata ($DTAs$) and deterministic hybrid automata ($DHAs$) is NP-complete. The problem of identifying approximately small DTA and DHA is also NP-complete.*

Proof. Assume that the class of deterministic timed automata C_{DTA} is given. Following Definition 4, Definition 5 and Observation 1, it is possible to represent the class of deterministic hybrid automata C_{DHA} using the class of C_{DTA}:

$$C_{DHA} = \{(S, s_0, F, \Sigma, T, \Delta, X, \Theta) \mid \exists(S, s_0, F, \Sigma, T, \Delta, X, \Theta') \in C_{DTA} : \Theta \supseteq \Theta'\}$$

where $\Theta' = \emptyset$. Assuming the class of deterministic finite automata C_{DFA} is also given, from Definition 5, Definition 6 and Observation 2 it follows:

$$C_{DTA} = \{(S, s_0, F, \Sigma, T, \Delta, X) \mid \exists(S, s_0, F, \Sigma, T, \Delta_0, X_0) \in C_{DFA} : \Delta \supseteq \Delta_0\, X \supseteq X_0\}$$

with $\Delta_0 = \emptyset$ and $X_0 = \emptyset$. Combining these relations, the class C_{DHA} can be decomposed in the following way

$$C_{DHA} = \{(S, s_0, F, \Sigma, T, \Delta, X, \Theta) \mid \exists(S, s_0, F, \Sigma, T, \Delta_0, X_0, \Theta') \in C_{DFA} : \Theta \supseteq \Theta', \Delta \supseteq \Delta_0, X \supseteq X_0\}.$$

Results from Theorem 1 and Theorem 2 for C_{DFA} now easily generalize to DTAs and DHAs. This result normally holds for all subclasses of DTAs and DHAs, including 1-DTAs and 1-DHAs. □

These findings do not look promising for automatic identification of deterministic hybrid automata from given finite data. However, research in a learning framework called the *identification in the limit* [Gol67] offered some interesting results.

3.4 Identification in the Limit

In this section, a formal definition of identification in the limit is provided. The known results that DFAs can be learned under this definition from informant and not from text is cited. This negative result normally generalizes to the classes of DTAs and DHAs (Corollary 1).

In the framework of identification in the limit [Gol67], the learning algorithm is not given predefined fixed set of learning examples, but more examples can be provided if necessary. This condition is satisfied in the application area of modeling already running hybrid production systems, where more data can be logged when needed.

For a better understanding of system variables and learning examples that represent the input to the learning process, we first give a general definition of an infinite sequence of learning examples.

Definition 9 (Infinite Sequence of Learning Examples). Let t denote the time, q_k $k = 1, ..., d$ be discrete binary signals, u_p $p = 1, ..., c$ be continuous input signals, y_r $r = 1, ..., o$ be continuous output signals. An infinite sequence of learning examples is a sequence:

$$\mathcal{D} = \{D_1, D_2, ..., D_n, ...\},$$

where each example D_i is distinct and represents a matrix of values:

$$D_i = \begin{bmatrix} t_1^i & q_{1,1}^i & \cdots & q_{d,1}^i & u_{1,1}^i & \cdots & u_{c,1}^i & y_{1,1}^i & \cdots & y_{o,1}^i \\ t_2^i & q_{1,2}^i & \cdots & q_{d,2}^i & u_{1,2}^i & \cdots & u_{c,2}^i & y_{1,2}^i & \cdots & y_{o,2}^i \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ t_l^i & q_{1,l}^i & \cdots & q_{d,l}^i & u_{1,l}^i & \cdots & u_{c,l}^i & y_{1,l}^i & \cdots & y_{o,l}^i \end{bmatrix},$$

where $j = 1, ..., l$, $t_j^i \in \mathbb{N}$ are the time stamps, $k = 1, ..., d$ and $j = 1, ..., l$, $q_{k,j}^i \in \{0, 1\}$ are the values of discrete binary signals, $p = 1, ..., c$ and $j = 1, ..., l$, $u_{p,j}^i \in \mathbb{R}$ are the values of continuous input signals, and $r = 1, ..., o$ and $j = 1, ..., l$, $y_{r,j}^i \in \mathbb{R}$ are the values of continuous output signals. The number of the time stamps, the number of values of every discrete signal q_k, the number of values of every continuous input signal u_p, and the number of values of every continuous output signal y_r in the example D_i is the same and equals to l.

For clarity, we give the following example.

Example 1. *A learning example D_i for $l = 4$ (number of values of every signal), $d = 2$ (two discrete binary signals), $c = 2$ (two continuous input signals), and $o = 1$ (one continuous output signal) could be given as:*

$$D_i = \begin{bmatrix} 1 & 1 & 0 & 1.3 & 7.4 & 10.4 \\ 2 & 0 & 0 & 1.5 & 7.2 & 11.7 \\ 3 & 0 & 1 & 1.8 & 7.1 & 12.2 \\ 4 & 0 & 1 & 2.1 & 6.7 & 13.3 \end{bmatrix},$$

where according to Definition 9 it follows: $[t_1^i, t_2^i, t_3^i, t_4^i] = [1, 2, 3, 4]$ are the values of the time stamp t, $[q_{1,1}^i, q_{1,2}^i, q_{1,3}^i, q_{1,4}^i] = [1, 0, 0, 0]$ are the values of the discrete binary signal q_1, $[q_{2,1}^i, q_{2,2}^i, q_{2,3}^i, q_{2,4}^i] = [0, 0, 1, 1]$ are the values of the discrete binary signal q_2, $[u_{1,1}^i, u_{1,2}^i, u_{1,3}^i, u_{1,4}^i] = [1.3, 1.5, 1.8, 2.1]$ are the values of the continuous input signal u_1, $[u_{2,1}^i, u_{2,2}^i, u_{2,3}^i, u_{2,4}^i] = [7.4, 7.2, 7.1, 6.7]$ are the values of the continuous input signal u_2, and $[y_{1,1}^i, y_{1,2}^i, y_{1,3}^i, y_{1,4}^i] = [10.4, 11.7, 12.2, 13.3]$ are the values of the continuous output signal y_1. In this example, the time between successive samples is incremented by one. However, this is not the case in general. Depending on the data acquisition mechanism and its precision, the timing between any two successive samples does not have to be equal. Moreover, this timing can be recorded in various units, such as seconds or milliseconds, which can always be represented as positive integers in order to satisfy Definition 4.

Please note that the learning example $D_i \in \mathcal{D}$ represents the string of symbols $z \in \Sigma^*$ given in the previous section. As mentioned before, those examples that are accepted by the automaton A are called positive examples for that automaton and they constitute the set $\mathcal{D}_+ \subseteq L_A$. All rejected examples belong to a set of negative examples $\mathcal{D}_-$.

According to Gold, identification in the limit is defined as follows:

Definition 10 (Identification in the Limit [Gol67]). A class of automata C is identifiable in the limit by an algorithm Ψ, if, for any automaton $A \in C$ and for any infinite sequence of examples $(\mathcal{D}_+, \mathcal{D}_-)$ with $\mathcal{D}_+ = L_A$ and $\mathcal{D}_- = \overline{L_A}$, there is some automaton A' and a number $n \in \mathbb{N}$, such that for all $i \geq n$, Ψ on input of the first i examples from $(\mathcal{D}_+, \mathcal{D}_-)$ returns A' and moreover $L_{A'} = L_A$.

Successful identification in the limit from informant assumes the existence of an input characteristic sample set with which the identification algorithm returns the correct (target) automaton ([dlH97], [Gol78], [VdWW08]). This set is defined as follows:

Definition 11 (Characteristic Set [VdWW08]). A characteristic set $\mathcal{D}_{ca}$ of an automaton A for an identification algorithm Ψ is a finite set of examples ($\mathcal{D}_{ca+} \in L_A$, $\mathcal{D}_{ca-} \in \overline{L_A}$) such that:

- given $\mathcal{D}_{ca}$ as input, algorithm Ψ returns the correct (target) automaton A, i.e. Ψ returns the automaton A' such that $L_{A'} = L_A$,
- $\mathcal{D}_{ca}$ needs to be monotonous, i.e. if more correctly labeled examples are added to $\mathcal{D}_{ca}$ such that $\mathcal{D}_+ \supseteq \mathcal{D}_{ca+}$ and $\mathcal{D}_- \supseteq \mathcal{D}_{ca-}$, the algorithm Ψ still identifies the automaton A' using ($\mathcal{D}_+$, $\mathcal{D}_-$), such that $L_{A'} = L_A$.

The framework of identification in the limit poses the work-around for the negative results given by Theorem 1 and Theorem 2. In 1967, Gold gave and proved the following theorem:

Theorem 3 ([Gol67, Pit89]). *DFAs are identifiable in the limit, and are not identifiable in the limit from text only.*

Proof (Sketch). In proving the first part of the theorem, Gold uses the technique called the *identification by enumeration* that always outputs the smallest automaton consistent with the so-far received examples. Although DFAs are identified in the limit, this technique is computationally inefficient [Pit89]. When it comes to identification of DFAs in the limit from text, Gold shows that this is not possible due to the following. In general, DFAs belong to a class of languages that contain all finite and at least one infinite language. Gold constructs a text for learning such an infinite language in the limit. This text constantly repeats a text for learning some of the finite languages from the same class. Since the repetitions can occur an infinite number of times (because the language is infinite), the identification algorithm mistakenly outputs the finite language indefinitely long. □

By following the decomposition given in the proof of Proposition 1, we can generalize the negative result of the previous theorem to the classes of DTAs and DHAs.

Corollary 1. *DTAs and DHAs are not identifiable in the limit from positive examples only.*

For this reason we have restricted our analysis of DHA identifiability in this chapter to informant identification only.

3.5 Polynomial Identification in the Limit

After obtaining the positive result for identifying DFAs in the limit from informant, the natural question that appeared is: *Could DFAs be identified in the limit from informant in polynomial time from data of some reasonable (polynomial) size*? Since this question has the great impact on learning DTAs and DHAs, it has been studied in this section. Out major result, that DHAs with one clock are polynomially identifiable in the limit from informant, is given by Theorem 8.

Definition 12 (Polynomial Identification in the Limit [dlH97, VdWW08]). A class of automata C is polynomially identifiable in the limit iff there exist two polynomials $PolyP$, $PolyQ$ and an algorithm Ψ such that:

- given an input data sample ($\mathcal{D}_+$, $\mathcal{D}_-$) of size[5] n, the algorithm Ψ returns the automaton $A \in C$ consistent with ($\mathcal{D}_+$, $\mathcal{D}_-$) in time bounded by $PolyP(n)$,
- for each automaton $A \in C$ of size $|A|$ (number of states), there exists a characteristic set ($\mathcal{D}_{ca+}$, $\mathcal{D}_{ca-}$) of size bounded by $PolyQ(|A|)$ for which, on data ($\mathcal{D}_+$, $\mathcal{D}_-$), if $\mathcal{D}_+ \supseteq \mathcal{D}_{ca+}$ and $\mathcal{D}_- \supseteq \mathcal{D}_{ca-}$, Ψ returns A' such that $L_{A'} = L_A$.

It is clear that if the given data ($\mathcal{D}_+$, $\mathcal{D}_-$) do not include a characteristic set ($\mathcal{D}_{ca+}$, $\mathcal{D}_{ca-}$) of size polynomial in the size of the automaton $A \in C$, the class C is not polynomially identifiable in the limit. If a class of (non-stochastic) deterministic automata C is polynomially identifiable in the limit, we say that it is identifiable from polynomial data in polynomial time. The existing results for the polynomial identification in the limit are now recalled.

Theorem 4 ([dlH97]). *If P $\neq$ NP, the class of non-deterministic finite state automata (NFAs) is not polynomially identifiable in the limit.*

Proof (Sketch). The proof builds up on a result presented in [GJ90] that given two NFAs, the problem of determining if they represent the same language (if they are equivalent) is co-NP-complete. This is the case even if the size of their alphabets is one ($|\Sigma| = 1$). From this result it follows that if the size of given two NFAs is smaller than some number $|A|$, there is no polynomial $Poly(|A|)$ that bounds the size of the input sample needed for testing the equivalence. According to Definition 12, the existence of such polynomial is required for polynomial identification in the limit. However, if the size of NFAs is exactly $Poly(|A|)$, the NFA equivalence problem would be in P (contradiction to the result of [GJ90]). Assuming P $\neq$ NP, it follows that NFAs are not polynomially identifiable in the limit. □

Again by following decomposition given in the proof of Proposition 1, this negative result easily generalizes to the *Non-deterministic Timed Automaton* (NTA) and *Non-deterministic Hybrid Automaton* (NHA). That is the reason why we restricted the analysis only to deterministic automata.

Corollary 2. *If P $\neq$ NP, the classes of NTAs and NHAs are not polynomially identifiable in the limit.*

The positive result for identification in the limit of DFAs given in Theorem 3 was further extended by E. M. Gold to account for the polynomial learning time.

Theorem 5 ([Gol78]). *The class of deterministic finite automata (DFAs) is polynomially identifiable in the limit.*

Proof (Sketch). Gold gives a DFA identification algorithm that has the following properties:

- feasible (resulting automaton consistent with given data ($\mathcal{D}_+$, $\mathcal{D}_-$)),
- identifies consistent automaton A using data ($\mathcal{D}_+$, $\mathcal{D}_-$) where $\mathcal{D}_+ \supseteq \mathcal{D}_{ca+}$ and $\mathcal{D}_- \supseteq \mathcal{D}_{ca-}$, and ($\mathcal{D}_{ca+}$, $\mathcal{D}_{ca-}$) is the characteristic set,

[5] The sample size denotes here the total sum of lengths of the examples. Please note that the sample size can sometimes also denote the number of learning examples [dlH06].

- computes in time polynomial in the size of learning data ($\mathcal{D}_+$, $\mathcal{D}_-$)),
- the size of the characteristic set is polynomial in the size of identified automaton.

These properties are satisfied using the *Timid State Characterization* algorithm, where the automaton transitions are identified using state characterization matrix (recall proof of Theorem 1). The algorithm ensures consistency of the automaton only if enough data (characteristic set) is available. In [OG92] yet another algorithm, called *Regular Positive and Negative Inference* (RPNI) was given that polynomially identifies DFAs in the limit by the means of state merging. □

In [VdWW08] Verwer investigated if this positive result for polynomial identification in the limit of DFAs could generalize to the class of deterministic timed automata (DTAs). His findings are summarized as follows:

Theorem 6 ([VdWW08]). *The class of deterministic timed automata ($DTAs$) is not polynomially identifiable in the limit.*

Proof (Sketch). In order to be polynomially identifiable in the limit, a class of automata C must have the property of polynomial distinguishability. This property means that given any two automata $A_1 \in C$ and $A_2 \in C$, such that $L_{A_1} \neq L_{A_2}$, a polynomial $Poly$ must exist that bounds the length of an example (string), which belongs to one automaton, and not to the other. DTAs with at least two clocks ($|X| = 2$, recall Definition 4) are not polynomially reachable, i.e. the examples of the length that is exponential in the size of the automaton may be needed in order to reach some of its states[6]. This is of course valid for the entire class of DTAs. Since they are not polynomially reachable, these automata are not polynomially distinguishable. The automaton characteristic set contains examples that cannot be bounded by a polynomial function, and thus, by Definition 12, DTAs are not polynomially identifiable in the limit. □

This again generalizes to deterministic hybrid automata (DHAs).

Corollary 3. *The class of deterministic hybrid automata (DHAs) is not polynomially identifiable in the limit.*

There is, however, one subclass of DTAs that can be identified from polynomial data in polynomial time. This class is called *One-clock Deterministic Timed Automata* (1-DTAs), and it represents the class of DTAs with a single clock, i.e. $|X| = 1$ (recall Definition 5).

Theorem 7 ([VdWW09]). *The class of one-clock deterministic timed automata (1-DTAs) is polynomially identifiable in the limit.*

Proof (Sketch). The proof is not trivial and it consists of two parts given in papers [VdWW08] and [VdWW09]. We give only the main elements here. The first paper shows that 1-DTAs are, in contrast to other DTAs, polynomially reachable, which is a requirement for polynomial distinguishability. This was proven by observing that the shortest example that ends in some state $s \in S$ can cause at most $|S|$ resets of a clock $x \in X$, and that this example can visit each state at most $|S|$ times. Thus, its length is bounded by $|S| \times |S|$. The length of the shortest example that belongs to a symmetric difference of two 1-DTAs is also bounded by polynomial, which makes this class polynomially distinguishable. The paper [VdWW09] presents an identification algorithm ID_1DTA that satisfies the following properties:

[6] Verwer gives an example of a DTA that is not polynomially reachable in [VdWW08].

- identification of a single transition requires time polynomial in the size of the input data,
- the number of transitions to be identified is polynomial in the size of the input data,
- for each transition there exists a characteristic set of size polynomial in the size of the smallest 1-DTA consistent with the input examples,
- the number of transitions to be identified is polynomial in the size of the smallest 1-DTA consistent with the input examples.

By Definition 12 this algorithm identifies 1-DTAs in polynomial time from polynomial data, thus 1-DTAs are polynomially identifiable in the limit. □

Our goal now is to extend this positive result to the subclass of deterministic hybrid automata (DHAs), namely the *One-clock Deterministic Hybrid Automata* (1-DHAs). A specific characteristic of 1-DHAs that distinguishes them from 1-DTAs is the ability to model the continuous dynamics by learning Θ functions using continuous data. In order for 1-DHAs to be polynomially identifiable in the limit in the sense of Definition 12, their Θ functions have to be learnable in polynomial time. This requirement is given by the following lemma.

Lemma 1. *A method $\mathcal{M}$ exists that learns functions $\theta_s \in \Theta, \forall s \in S$ of any DHA (including 1-DHAs) within predefined marginal error ε in polynomial time.*

Proof. According to the lemma statement, a method $\mathcal{M}$ has to have the following properties:

$$error(\mathcal{M}(\theta_s)) \leq \varepsilon \text{ and}$$

$$runtime(\mathcal{M}(\theta_s)) \in O(Poly)$$

for some polynomial $Poly$.

For proving the first property it is enough to recall the *Universal approximation theorem* given by G. Cybenko in 1989 [Cyb89]. By this theorem, a feed-forward artificial neural network with a finite number of hidden neurons in a single hidden layer and continuous sigmoid (as well as other more general) activation functions is an universal approximator of continuous functions.

Now let method $\mathcal{M}$ be approximator from the Universal approximation theorem. Neural network time complexity for a single approximation is $O(kmME)$, with k data points for approximation, m predictors, M hidden neurons and E training epochs [HTF08]. In the case of DHA, predictors are all continuous input variables U and the time t, thus the number of predictors is $m = |U| + 1$, and there are $|\Theta|$ functions to approximate. Therefore, the overall time complexity of the method $\mathcal{M}$ is $O(k|U|ME|\Theta|)$. This completes the proof. □

The value of the predefined marginal error ε depends on the application area. In safety-related tasks, it is important that θ functions approximate the continuous dynamics in the system as good as possible, i.e. with very small error (e.g. less than 5%). Information about allowed ε values for different signals come both from the analysis of a system and expert knowledge.

Theorem 8. *The class of one-clock deterministic hybrid automata (1-DHAs) is polynomially identifiable in the limit.*

Proof. Polynomial identification in the limit of 1-DTAs has been given by Theorem 7. In the proof of Proposition 1, the class of deterministic timed automata (DTAs) is shown as a subclass of deterministic hybrid automata (DHAs):

$$C_{DHA} = \{(S, s_0, F, \Sigma, T, \Delta, X, \Theta) \mid \exists(S, s_0, F, \Sigma, T, \Delta, X, \Theta') \in C_{DTA} : \Theta \supseteq \Theta'\}$$

where $\Theta' = \emptyset$. The same relationship holds for the subclasses with only one clock, i.e. when $|X| = 1$:

$$C_{1-DHA} = \{(S, s_0, F, \Sigma, T, \Delta, \{x\}, \Theta) \mid \exists(S, s_0, F, \Sigma, T, \Delta, \{x\}, \Theta') \in C_{1-DTA} : \Theta \supseteq \Theta'\}$$

where $\Theta' = \emptyset$. If Θ functions of 1-DHA would not have to be learned, we would essentially have to learn 1-DTA, which, according to Theorem 7, are polynomially identifiable in the limit. Since learning Θ functions is also possible in polynomial time (Lemma 1), 1-DHAs are also polynomially learnable in the limit. This completes the proof. □

Corollary 4. *1-DHAs are polynomially identifiable in the limit when their continuous output signals $y \in Y$ are approximated by functions $\theta_s \in \Theta$, $\forall s \in S$ within marginal error ε using multiple linear regression with ordinary least squares method for parameter estimation.*

Proof. As the corollary states, for multiple linear regression with ordinary least squares (OLS) estimation it holds:

$$error(OLS(\theta_s)) \leq \varepsilon, \forall \theta_s \in \Theta, \forall s \in S.$$

The time complexity of the least squares method that learns a single function is $O(km^2)$ [HTF08] where k is the number of data points used for approximation and m is the number of parameters to be estimated. In order to learn a regression function $\theta_s \in \Theta$ for all states $s \in S$ of the automaton A, the least squares estimation needs to be performed $|S|$ times. This increases the approximation time complexity to $O(km^2|S|)$. Further, in the worst case there is no expert knowledge about which output signals from the set Y are relevant for which states. In this case all of them need to be approximated in every single state. This gives the final time complexity of $O(km^2|S||Y|)$. The approximation is polynomial in both the number of output signals $|Y|$ and the number of states $|S|$, thus 1-DHAs are polynomially identifiable in the limit when multiple linear regression with ordinary least squares method is used for parameter estimation. □

Observation 3. *When multiple linear regression uses only linear terms, the number of parameters that need to be estimated in the worst case is:*

$$m = |U| + 2,$$

where $|U|$ is the number of input signals (Definition 4), and additional two parameters are the constant and the time coefficient. For this scenario, the total approximation time complexity is $O(k|U|^2|\Theta|)$, which can be written as $O(k|U|^2|S||Y|)$ (knowing that $|\Theta| = |S||Y|$). The approximation is polynomial in both the number of input signals $|U|$ and the number of functions $|\Theta|$ that approximate output signals. Therefore, 1-DHAs are polynomially identifiable in the limit when multiple linear regression with linear terms is used for parameter estimation.

Although this theoretical result is positive, please note that the polynomial identification in the limit of 1-DHAs remains hard task in many practical cases. For example, even when a simple regression method such as multiple linear regression is used, it is not always easy to satisfy all multiple linear regression assumptions (like e.g. normal distribution of multiple regression residuals [MC04]). Another issue is the identification time. To illustrate this, it is enough to recall that the number of parameters to be estimated in ordinary least square estimation is $m = |U| + 2$ for each approximation. In a complex process plant, number of input continuous signals $|U|$ of a modeled component can be large. As the time complexity of multiple linear regression with ordinary least squares estimation is quadratic in $|U|$, the approximation runtime could still be long.

3.6 Conclusion

This chapter gave an overview of complexity results for the identification of deterministic finite automata (DFAs) and deterministic timed automata (DTAs) from data, and generalized them to deterministic hybrid automata (DHAs). Due to negative results in text identification (i.e. from positive examples only) of (non-stochastic) deterministic finite automata, the analysis is focused on informant identification (both positive and negative examples are available). The problem of identifying a DFA with k states is proven to be NP-complete. Furthermore, this DFA cannot be approximated within any polynomial in k. This holds for both DTAs and DHAs. However, in the learning framework of identification in the limit, DFAs can be identified successfully, and moreover in polynomial time using polynomial data. This result generalizes to a subclass of DTAs, called one-clock deterministic timed automata (1-DTAs). In Theorem 8, we extended this finding to the one-clock deterministic hybrid automata (1-DHAs). The algorithm ID_1DTA for learning 1-DTAs can be modified to approximate the continuous dynamics in the states in order to learn 1-DHAs polynomially in the limit from informant.

Despite this positive theoretical result, two major obstacles exist in practice. Our ultimate goal is to identify behavior models of hybrid systems. First, probabilistic behavior in such systems is common, and it cannot be modeled using (non-stochastic) deterministic hybrid automata for which a positive identification result is given. Second, learning examples that can be logged in hybrid (production) systems are dominantly positive [Ang88a, CO99] (i.e. measurements come from normal system behavior), thus text identification environment is a prerequisite in this application area.

For these reasons, the following chapter analyses complexity of identifying stochastic deterministic hybrid automata from text.

Chapter 4

Complexity of Identifying Stochastic Deterministic Automata

The goal of this chapter is to provide the background knowledge relevant for the identification of *Stochastic Deterministic Hybrid Automata* (SDHAs), which are capable of representing real-world hybrid production systems. In particular, the key points in this chapter are:

- basic notations, formal definitions of three stochastic deterministic automata classes (SDFAs, SDTAs and SDHAs) and identification settings are given,
- an existing negative result for identification of SDFAs (Theorem 9) from a reasonable number of positive examples is cited (i.e. from data of the size that is polynomially linked with the size of the target automaton) and generalized to stochastic deterministic timed and hybrid automata with one clock that counts the relative time between successive changes in a system[1],
- existing positive results are presented for identifying SDFAs (Theorem 10) and SDTAs with one clock (Theorem 11) from positive examples in a reasonable time (i.e. in time that is polynomially linked with the size of the input data).

We will relate to these results in Chapter 6, where we present our algorithm for learning SDHAs with one clock and prove its convergence and runtime properties. We emphasize that this chapter deals with learning stochastic deterministic automata from text, i.e. from positive learning examples only.

[1] Whenever we refer to one clock automata in this thesis, we assume the clock uses the relative timing. Such automata with one clock that resets at every transition are often called *real-time automata* (e.g. [Ver10]).

4.1 Introduction

This section introduces learning stochastic finite automata from text and explains the structure of this chapter.

Despite the positive identification result for non-stochastic one-clock deterministic hybrid automata given in the previous chapter, two major problems remained: (*i*) non-stochastic automata cannot model probabilistic behavior, and (*ii*) in the real-world, available data for learning contain dominantly positive examples [Ang88a, CO99]. These constraints, imposed by our application area, make it an imperative to use the text identification environment for learning stochastic automata (i.e. to learn from positive examples only).

The first challenge to overcome was Gold's result given in [Gol67], that even the simplest deterministic automata, such as DFAs, cannot be identified in the limit from text. However, Angluin shows in [Ang88a] that the assumption of stochastic behavior of the underlying process that generates the sequence of positive learning examples has huge implications on automata identification. Statistical regularity of data samples, generated according to the same probability distribution, is able to compensate for the lack of negative data. In this chapter, we present the known results for text identification of *Stochastic Deterministic Finite Automata* (SDFAs) and *Probabilistic Deterministic Real-Time Automata* (PDRTAs) in order to extend them to the class of *One-clock Stochastic Deterministic Hybrid Automata* (1-SDHAs) in Chapter 6. Please note that some authors use the term *probabilistic*, rather than *stochastic*. We decided to follow the original nomenclature of automata classes from their corresponding papers. The reader can find an extensive survey of stochastic automata in [VTdlH$^+$05a].

The chapter is organized as follows. In Section 4.2, these automata classes are formally defined and some important background information is given. The extension of Gold's identification framework [Gol67] to the stochastic automata classes is called *identification in the limit with probability one* [dlHO04]. It is defined in Section 4.3 together with the strong and weak polynomial identification criteria. Well-known negative results concerning the former are summarized in Section 4.4 (Theorem 9) and extended to the classes of PDRTAs and 1-SDHAs (Corollary 5). The latter criteria are analyzed in Section 4.5, where the well-known positive results for SDFAs and PDRTAs are given (Theorem 10 and Theorem 11, respectively). We use these results in Chapter 6, where we present our algorithm that learns SDHAs with one clock and prove its properties.

4.2 Notations and Automata Definitions

In this section, three classes of stochastic finite automata are formally defined and their relations are established. In addition, the reader is introduced to the important notations used throughout the thesis.

In *informant identification* the learning data come in a form of sets of positive and negative examples, usually denoted by $(\mathcal{D}_+, \mathcal{D}_-)$. The subject of this chapter is the *text identification* environment, where $\mathcal{D}_- = \emptyset$. For simplicity, the set of available positive examples will be denoted by $\mathcal{D}$ instead of $\mathcal{D}_+$. In the following, we explain used notations that are similar to [CO99].

An alphabet Σ is a finite set of symbols. Symbols are denoted by letters a, b, c. Strings of symbols represent the examples (words), usually marked by letters u, v,...,z. An empty string is denoted by λ. Σ^* is a set of all finite strings of symbols from Σ. A stochastic language L represents a probability density function over Σ^*. An automaton A defines a stochastic language L_A. A probability of a string $z \in \Sigma^*$ in the language L_A is denoted by $p(z|L_A)$. The sum of probabilities of all strings in Σ^* that belong to a language L_A equals to one [dlH10, HMU01], i.e. $\sum_{z \in \Sigma^*} p(z|L_A) = 1$. For a given data $\mathcal{D}$ that belong to an automaton language L_A over Σ, it holds $\mathcal{D} \subseteq \Sigma^*$. Two automata A and A' from some class of stochastic automata C are equivalent, if they define the same stochastic language, i.e. if $L_A = L_{A'}$. Two stochastic languages are equal if they include the same strings and if probability of every string is equal:

$$L_A = L_{A'} \iff \forall z \in \Sigma^* : p(z|L_A) = p(z|L_{A'}).$$

Definition 13 (Stochastic Deterministic Hybrid Automaton). A stochastic deterministic hybrid automaton (SDHA) is a tuple $A = (S, s_0, \Sigma, T, \Delta, P, X, \Theta)$, where

- S is a finite set of states and $s_0 \in S$ is the initial state.
- Σ is a finite set called the alphabet. Its elements are symbols that trigger transitions between the states.
- $T \subseteq S \times \Sigma \times S \times \Delta$ is a finite set of transitions. A transition $\tau \in T$ is a tuple (s, a, s', δ), where $s, s' \in S$ are the source and destination states, $a \in \Sigma$ is the trigger symbol, and $\delta \in \Delta$ is the timing constraint.
- $\Delta \subseteq \{\delta = [t_1, t_2] : t_1, t_2 \in \mathbb{N}\}$ is a finite set of transition timing constraints. Constraints $\delta \in \Delta$ model the time spent in a state before the transition takes place.
- P is a set of probability functions with the elements $p : S \times (\Sigma \cup \{\lambda\}) \times S \times \Delta \to \mathbb{Q} \cap [0, 1]$. P includes both transition probabilities and probabilities of a string ending in a state.
- X is a finite set of clocks that record the continuous time evolution. The valuation of the clock $x \in X$ is defined by $v_t(x) : X \to \mathbb{N}$.
- Θ is a finite set of functions with elements $\theta_s : \mathbb{R}^n \to \mathbb{R}^m, \forall s \in S, n, m \in \mathbb{N}$. I.e. $y = \theta_s(t, u)$ is the function computing the value changes of the output signals $y \in Y$ within state s based on the time t and values of continuous input signals $u \in U$. With Q we denote a set of discrete signals.

As in the case of DHAs (see Definition 4), θ functions of SDHAs can also be learned using a range of regression methods (e.g. multiple linear regression).

Please note, that in contrast to non-stochastic deterministic automata (such as DFA), stochastic deterministic automata do not contain final (accepting) states. Learning examples (also called *strings* or *words* in the literature) could end in arbitrary states, and moreover these examples are strictly positive. Therefore, the notation of final states is here not required. Furthermore, the analysis is in this chapter restricted to those stochastic automata that are deterministic. A deterministic automaton is one that does not include two transitions from the same state, which have the same triggering symbol and overlapping timing constraints. As in [HT00], only rational probabilities will be used. The reason is the way the probabilities are estimated from learning examples, i.e. as ratio of the number of examples that leave the automaton state (or stay in that state) and the number of examples that attain the state.

Verwer proved in [VdWW08] that the class of non-stochastic deterministic timed automata (DTAs) with more than one clock that counts time, cannot be efficiently

identified (i.e. in polynomial time from polynomial data, see Theorem 6 in Section 3.5). This negative result naturally generalizes to all deterministic automata that model time with clocks. The same argument holds also for stochastic automata [Ver10]. Therefore, the attention is directed here to the subclass of DHAs, namely a *One-clock Stochastic Deterministic Hybrid Automaton*. It uses relative timing, i.e. the clock counts the time between two successive events. Whenever a new event is observed, the clock is reset. To ease readability, this automaton is separately defined.

Definition 14 (One-Clock Stochastic Deterministic Hybrid Automaton). A one-clock stochastic deterministic hybrid automaton (1-SDHA) is a SDHA that uses a single clock to record the continuous time evolution, i.e. $|X| = 1$.

An example of a 1-SDHA is given in Figure 4.1. According to Definition 13 and Definition 14, it follows: $s_0, s_1, s_2 \in S$, $a, b \in \Sigma$, $\delta_1, \delta_2 \in \Delta$, and $\theta_{s_0}, \theta_{s_1}, \theta_{s_2} \in \Theta$. Assume the system is in the state s_0 where its continuous dynamics is defined by the function θ_{s_0}. If the symbol a is observed at the time instance that satisfies the constraint δ_1, a transition occurs to the state s_1. In the recorded logs of a system, such transition was triggered by 70% of the examples, giving $p(s_0, a, s_1, \delta_1) = 0.7$. The probability of the other transition is $p(s_0, b, s_2, \delta_2) = 0.2$. The probability of staying in the state s_0 is denoted by $p(s_0, \lambda, s_0, \delta_0)$, where λ is an empty string and $\delta_0 = [0, 0]$ is a notation used for the timings constraint in the ending state probability. It is important to make clear that λ does not trigger any transitions from the state s_0. For simplicity, we will denote the ending state probability of the state s_0 with $p(s_0)$ instead of $p(s_0, \lambda, s_0, \delta_0)$. Although we conveniently write probabilities as real numbers, please keep in mind that we basically use only rational probabilities, i.e. aforementioned probabilities can be as well written as $p(s_0, a, s_1, \delta_1) = \frac{7}{10}$, $p(s_0, b, s_2, \delta_2) = \frac{1}{5}$ and $p(s_0) = \frac{1}{10}$.

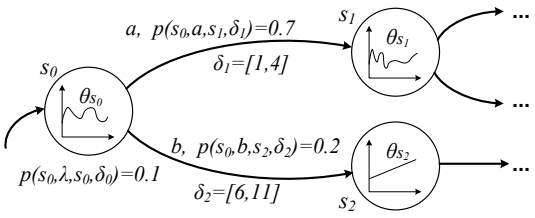


Fig. 4.1 An example of one-clock stochastic deterministic hybrid automaton.

One-clock Stochastic Deterministic Timed Automaton (1-SDTA) has been defined by Verwer in [Ver10] and called *Probabilistic Deterministic Real-Time Automaton* (PDRTA). As stated before, in this chapter we use the original notations, thus the name PDRTA will be used. In order to present the identification results for PDRTAs and to place them in the broader picture of identifying stochastic deterministic automata, their definition is given in the following.

Definition 15 (Probabilistic Deterministic Real-Time Automaton [Ver10]). A probabilistic deterministic real-time automaton (PDRTA) is an ordered list of elements $A = (S, s_0, \Sigma, T, H, P_\Sigma, P_t)$, where

- S is a finite set of states and $s_0 \in S$ is the initial state.

- Σ is a finite set called the alphabet. Its elements are symbols that trigger transitions between the states.
- T is a finite set of transitions. A transition $\tau \in T$ is a tuple $(s, a, s', [t_1, t_2])$, where $s, s' \in S$ are the source and destination states, $a \in \Sigma$ is the trigger symbol, and $[t_1, t_2]$, $t_1, t_2 \in \mathbb{N}$ is the time interval.
- H is a finite set of bins (time intervals) $h = [t_1, t_2]$, $t_1, t_2 \in \mathbb{N}$, known as histogram.
- $P_\Sigma = \{p(s, a, *) \mid a \in \Sigma, s \in S\}$ is a finite set of symbol probabilities. For every state $s \in S$, it holds $\sum_{a \in \Sigma} p(s, a, *) = 1$, where $*$ is an arbitrary state.
- $P_t = \{p(s, h, *) \mid h \in H, s \in S\}$ is a finite set of time-bin probabilities. For every state $s \in S$, it holds $\sum_{h \in H} p(s, h, *) = 1$, where $*$ is an arbitrary state.

Please note that PDRTA also has only one clock that counts relative time. An example of PDRTA is given in Figure 4.2.

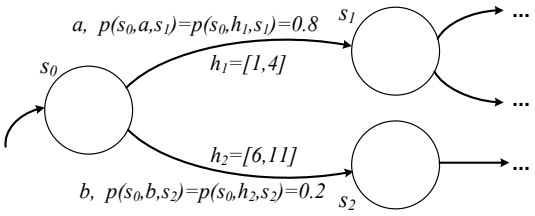


Fig. 4.2 An example of probabilistic deterministic real-time automaton.

In the following, a stochastic version of the well-known *Deterministic Finite Automaton* (DFA) is defined. It is called *Stochastic Deterministic Finite Automaton* (SDFA) and illustrated in Figure 4.3.

Definition 16 (Stochastic Deterministic Finite Automaton [CO99, HT00]). A stochastic deterministic finite automaton (SDFA) is a tuple $A = (S, s_0, \Sigma, T, P)$, where

- S is a finite set of states and $s_0 \in S$ is the initial state.
- Σ is a finite set called the alphabet. Its elements are symbols that trigger transitions between the states.
- $T \subseteq S \times \Sigma \times S$ is a finite set of transitions. A transition $\tau \in T$ is a tuple (s, a, s'), where $s, s' \in S$ are the source and destination states, and $a \in \Sigma$ is the trigger symbol[2].
- P is a set of probability functions with the elements $p : S \times (\Sigma \cup \{\lambda\}) \times S \to \mathbb{Q} \cap [0, 1]$. P includes both transition probabilities and probabilities of a string ending in a state.

Observation 4. *A stochastic deterministic finite automaton (SDFA) is a probabilistic deterministic real-time automaton (PDRTA)* $A = (S, s_0, \Sigma, T, H, P_\Sigma, P_t)$*, with* $H = \emptyset$ *(no time intervals are associated with transitions),* $P_t = \emptyset$*, and a final set of symbol probabilities* P_Σ *extended with an empty string to obtain* $P_{\Sigma \cup \{\lambda\}}$*.*

[2] Some authors give a set of transitions as a transition function $\tau : S \times \Sigma \to S$.

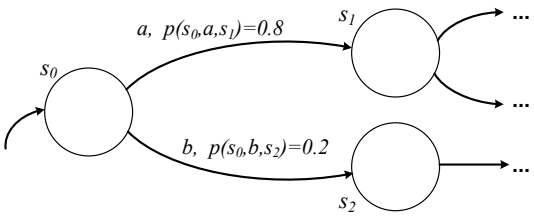


Fig. 4.3 An example of stochastic deterministic finite automaton.

Observation 5. *A stochastic deterministic finite automaton (SDFA) is a one-clock stochastic deterministic hybrid automaton (1-SDHA)* $A = (S, s_0, \Sigma, T, \Delta, P, X, \Theta)$, *with* $\Delta = \emptyset$, $X = \emptyset$ *and* $\Theta = \emptyset$.

4.3 Identification in the Limit with Probability One

Here we define *positive learning examples* and the *identification in the limit with probability one*, as well as weak and strong identification criteria.

This chapter deals with learning stochastic deterministic automata. As already stated, the goal is to present some well-known results for identification of SDFAs and PDRTAs, but also to answer the following open question: *Given any infinite sequence of positive examples (that eventually contains all of them) drawn according to some probability distribution, can a one-clock stochastic deterministic hybrid automaton that defines that distribution be automatically identified?* The general definition of an infinite sequence of learning examples is already given in Section 3.4 (Definition 9) and it is also applicable for the case of learning stochastic automata. Since we use only text for learning, we define positive learning examples.

Definition 17 (Positive Learning Examples). Learning examples from the infinite sequence:

$$\mathcal{D} = \{D_1, D_2, ..., D_n, ...\}$$

are denoted as positive with respect to the language L_A if:

$$\forall D_i \in \mathcal{D} : p(D_i|L_A) > 0,$$

i.e. if all examples are drawn according to the language L_A.

Every learning example $D_i \in \mathcal{D}$ corresponds to the string of symbols $z \in \Sigma^*$ explained in the previous section. These two notations are in this chapter used interchangeably.

In order to identify stochastic automata, the identification framework similar to Gold's *identification in the limit* [Gol67] for learning non-stochastic automata is used (see Definition 10). In the setting of learning stochastic automata, it is called *identification in the limit with probability one* [dlHO04]. The main idea is that as the sequence of positive examples used for learning increases, it gets unlikely that the identified empirical probability distribution (automaton) will be too far away from the theoretical (target) one [dlH10]. Therefore, the probability of converging to the target automaton after some finite time equals to one. Of course, the actual moment

of convergence cannot be known beforehand. Nevertheless, one can try to find an algorithm that identifies such automaton correctly, and moreover, in polynomial time. The following definition is given (based on [dlHO04]):

Definition 18 (Identification in the Limit with Probability One). A class of automata C is identifiable in the limit with probability one, if there exists an identification algorithm Ψ that given any automaton $A \in C$ and any infinite sequence of positive examples $\mathcal{D} = \{D_1, D_2, ..., D_n, ...\}$ for A (that eventually contains all examples from the language L_A defined by A) it holds:

$$\exists n \in \mathbb{N}, \forall i \geq n : p(L_{\Psi(\{D_1, D_2, ..., D_i\})} = L_A) = 1.$$

Please note that the identification takes place if and only if the examples from $\mathcal{D}$ are drawn according to the language (distribution) L_A that A defines (recall Definition 17).

This definition ensures that output automaton will eventually converge to the target automaton. However, it does not constrain the time and data size needed for convergence. For that reason, the following two definitions are provided (also based on [dlHO04]). The first one bounds the time, while the second bounds both the time and data size.

Definition 19 (Weak Polynomial Identification in the Limit with Probability One). A class of automata C is **weak polynomially identifiable** in the limit with probability one, if there exists an identification algorithm Ψ and a polynomial $PolyP$ for which for any automaton $A \in C$ and any infinite sequence of positive examples $\mathcal{D} = \{D_1, D_2, ..., D_n, ...\}$ drawn according to the language L_A that A defines:

- Ψ identifies A in the limit with probability one,
- Ψ works in time bounded by $PolyP(||\{D_1, D_2, ..., D_n\}||)$, where $n \in \mathbb{N}$ is the minimal number of positive examples from Definition 18 and $||\{D_1, D_2, ..., D_n\}||$ is the sum of lengths of the examples $D_i, i = 1, ..., n$.

Definition 20 (Strong Polynomial Identification in the Limit with Probability One). A class of automata C is **strong polynomially identifiable** in the limit with probability one, if there exists an identification algorithm Ψ and two polynomials $PolyP$ and $PolyQ$ for which for any automaton $A \in C$, and any $\gamma > 0$, and any infinite sequence of positive examples $\mathcal{D} = \{D_1, D_2, ..., D_n, ...\}$ drawn according to the language L_A that A defines:

- Ψ identifies A in the limit with probability one,
- Ψ works in time bounded by $PolyP(||\{D_1, D_2, ..., D_n\}||, \frac{1}{\gamma})$, where $n \in \mathbb{N}$ is the minimal number of positive examples from Definition 18 and $||\{D_1, D_2, ..., D_n\}||$ is the sum of lengths of the examples $D_i, i = 1, ..., n$,
- if $n \geq PolyQ(|A|, \frac{1}{\gamma})$, then $p(L_{\Psi(\{D_1, D_2, ..., D_n\})} = L_A) \geq 1 - \gamma$.

Variable γ represents the confidence parameter, i.e. it is a probability that data sampling (acquisition of the learning data) has gone wrong. It is intuitively clear that the smaller γ, the more data would be required for learning. The number $|A|$ is the size of the automaton A, i.e. the number of its states.

For easier reading, we introduce several abbreviations. Identification in the limit with probability one is abbreviated with 'IDLimitProb1'. For weak and strong polynomial IDLimitProb1, we will use 'WeakPolyIDLimitProb1' and 'StrongPolyIDLimitProb1' respectively. We emphasize that all three identification definitions relate to the identification from positive examples only, i.e. from text.

In order to place the identification of one-clock stochastic deterministic hybrid automata (1-SDHAs) in a broader landscape of learning stochastic automata from text, we give Table 4.1. It shows the learnability of different stochastic automata defined in the previous section, with regard to aforementioned identification definitions. It can be seen that results already exist for SDFAs and PDRTAs. In this thesis, we answer the open questions for PDRTAs and 1-SDHAs.

Table 4.1 A landscape of stochastic deterministic automata identifiability from text.

Identification definition	**Automata**		
	SDFAs	PDRTAs	1-SDHAs
IDLimitProb1	Yes	Yes	?
WeakPolyIDLimitProb1	Yes	Yes	?
StrongPolyIDLimitProb1	No	?	?

4.4 Strong Polynomial Criteria for Identification

In this section, we cite the negative StrongPolyIDLimitProb1 result for SDFAs (see Table 4.1) and generalize it to PDRTAs and 1-SDHAs.

Strong polynomial identification in the limit with probability one given by Definition 20, requires a successful convergence, polynomial runtime in the size of the input data, and moreover, identification from polynomial number of examples in the size of the target automaton. We recall aforementioned major negative StrongPolyIDLimitProb1 result in the following theorem (due to de la Higuera [dlHO04]).

Theorem 9 ([dlHO04]). *The class of SDFAs is not StrongPolyIDLimitProb1.*

Proof (Sketch). Probabilistic languages $A_1, A_2, ..., A_m$ (distributions, automata) are considered. Distinguishing with high probability that a string z belongs to one language and not the other requires already a dataset of the size exponential in m. The argument is that different distributions are all binomials. In order to distinguish each binomial with probability at least $1 - \gamma$ (see Definition 20), an exponential number of strings in m is needed. □

Following Observation 4 and Observation 5, this negative result generalizes to the classes of PDRTA and 1-SDHA.

Corollary 5. *PDRTAs and 1-SDHAs are not StrongPolyIDLimitProb1.*

Verwer argues without proof in [Ver10] that, in analogy to *non-stochastic One-clock Deterministic Timed Automata* (1-DTAs) [VdWW08], a sufficient input data (called a *characteristic set*) of polynomial size could also exist for learning PDRTAs.

If this were the case, then PDRTAs would be StrongPolyIDLimitProb1. Nevertheless, Theorem 9 and Corollary 5 reject this possibility. Of course, learning some specific subclass of PDRTAs (or 1-SDHAs) under StrongPolyIDLimitProb1 criteria could still be possible. Finding such a subclass has been left to future work, as the research presented in this thesis is focused on the algorithm that learns 1-SDHAs under WeakPolyIDLimitProb1 definition.

Due to negative results for StrongPolyIDLimitProb1, the following section focuses on WeakPolyIDLimitProb1, i.e. on the results in ensuring identification convergence and polynomial runtime of algorithms, which learn stochastic deterministic automata.

4.5 Weak Polynomial Criteria for Identification

This section presents known positive results concerning WeakPolyIDLimitProb1 and IDLimitProb1 of SDFAs and PDRTAs, i.e. 'Yes' answers from Table 4.1 are here addressed. It serves as basis for our own new results in identification of 1-SDHAs, which are given in Chapter 6.

According to Definition 19, two preconditions are required for WeakPolyIDLimit-Prob1: convergence of the learning algorithm to the target automaton (identifying structure and probabilities), and its polynomial runtime in the size of the input data.

4.5.1 Identification of SDFAs

De la Higuera has addressed the question of WeakPolyIDLimitProb1 of SDFAs [dlHO04]. He proves that SDFAs are WeakPolyIDLimitProb1, and also implicitly that SDFAs are IDLimitProb1. The following theorem gives this result.

Theorem 10 ([dlHO04]). *The class of SDFAs is WeakPolyIDLimitProb1.*

Proof (Sketch). The proof uses the Chebyshev distance measure between two distributions [APR02] (automata) A and A':

$$d_\infty(A, A') = \max_{z \in \Sigma^*} \mid p(z|A) - p(z|A') \mid .$$

Assuming the examples are randomly drawn according to distribution A, A' represents the empirical distribution built from a sample of the size n. Angluin [Ang88a] gives a measure $I(n) = \sqrt{6a(\log\ n)/n}$, $a > 1$, so that with probability one it holds:

$$d_\infty(A, A') \leq I(n).$$

Using a simple enumerative algorithm, a distribution is identified for which this inequality holds. As the size n of the sample grows, the empirical distribution A' comes closer to the target distribution A, which generated the learning data. Eventually, the convergence takes place (i.e. in the limit with probability one). To make the algorithm polynomial, the time is tracked that the algorithm is entitled to in order to remain polynomial, using the current examples. The algorithm runs as far as it can go within that time, and returns whatever solution it obtains. With the next example it gets more time, and continues the previous computation. Basically the order of the polynomial changes, but the algorithm remains polynomial. □

Corollary 6. *Previous theorem implicitly states that SDFAs are IDLimitProb1. The first part of the theorem proof is enough to prove this corollary.*

Two famous algorithms for learning SDFAs are already described in Section 2.4, namely the ALERGIA [CO94], and the *Minimal Divergence Inference* (MDI) algorithm [TDdlH00]. They identify SDFAs in the sense of WeakPolyIDLimitProb1 definition. The convergence of the ALERGIA algorithm has been formally proven [CO99, HT00].

4.5.2 Identification of PDRTAs

Identification of stochastic deterministic timed automata (SDTAs) was researched by Verwer [Ver10]. For a subclass called probabilistic deterministic real-time automata (PDRTA, see Definition 15), i.e. timed automata with only one clock that counts the time between consecutive events (relative timing), we recall the following theorem:

Theorem 11 ([Ver10]). *The class of PDRTAs is WeakPolyIDLimitProb1.*

Proof (Sketch). Verwer gives the algorithm *Real-Time Identification from Positive Data* (RTI+) [VdWW10] that identifies PDRTAs. It is based on an informant identification counterpart algorithm RTI (both positive and negative examples exist) that learns *Deterministic Real-Time Automata* (DRTAs)[3]. The RTI+ identifies both the automaton structure (states and transitions), as well as two probability distributions: probabilities of the events (symbols that trigger transitions), and time probabilities. These distributions are assumed to be independent. In addition to merging, a splitting step is also introduced. The decision whether to merge two states or split a single state is based on the likelihood-ratio test that gives the p-value. High p-value indicates that two states are similar and can be merged. Low p-value indicates that a state should be split. If a split of a state is found that results in a p-value lower than 0.05, the split with the lowest p-value is performed. If a merge is found with the p-value larger than 0.05, the merge with the largest p-value is performed. Alternatively, other statistical tests can be used, such as the Kolmogorov-Smirnov test and Fisher's method [Ver10]. Verwer shows that by increasing the amount of learning data, the p-value resulting from any used test converges to zero for two different states. Therefore, in the limit such states will never be merged (p-value greater than 0.05 is needed to make a merge). Although some extra splits are possible, they do not influence the correctness of the learned language, but only increase the model size. Statistical tests are computed in polynomial time for every state. In addition, the algorithm makes maximum $2n^2 + n$ iterations, where n is the size of the input sample. Thus, the runtime of the RTI+ algorithm is polynomial in the size of the input data. □

Corollary 7. *Previous theorem implicitly proves that PDRTAs are IDLimitProb1.*

Verwer argues that RTI+ could be adjusted to identify SDFAs [Ver10], if statistical tests would be modified (presumably to exclude statistical tests for time probabilities from consideration).

[3] DRTAs are basically 1-DTAs where the clock resets at every transition.

4.5.3 Identification of 1-SDHAs

Aforementioned positive results for weak polynomial identifiability of SDFAs and PDRTAs in the limit with probability one from text look promising for the prospects of 1-SDHA identification. And indeed, we have developed a 1-SDHA learning algorithm and proven its WeakPolyIDLimitProb1 properties. Due to their importance, but also logical belonging to the Algorithms part of this thesis (Part III), they are positioned in Chapter 6 that presents our major contributions.

4.6 Summary

In this chapter, we have analyzed the identifiability of stochastic deterministic automata from positive examples only. Three definitions are given for identification in the limit with probability one: convergence to the target automaton (IDLimitProb1), convergence in polynomial time (WeakPolyIDLimitProb1), and convergence in polynomial time and from data of polynomial size (StrongPolyIDLimitProb1). Well-known results are presented for stochastic deterministic finite automata (SDFAs, Theorem 9, Theorem 10, and Corollary 6) and probabilistic deterministic real-time automata (PDRTAs, Theorem 11 and Corollary 7).

We generalized the negative result of StrongPolyIDLimitProb1 for SDFAs, to the classes of PDRTAs and 1-SDHA (Corollary 5). Unfortunately, there is no polynomial that bounds the data size needed for identification of any of these automata classes. However, positive WeakPolyIDLimitProb1 results for SDFAs and PDRTAs provide motivation to investigate the learnability of 1-SDHAs under the same criteria. These results are presented in the Part III of this thesis.

Chapter 5

Polynomial Approximations of Stochastic Automata

This chapter presents the foundations and major results in finding polynomial approximations of stochastic automata. Such approximations represent an alternative to learning the exact automata in the strong sense (i.e. in polynomial time from data of polynomial size). As given in the previous chapter (Theorem 9 and Corollary 5), the considered classes of stochastic automata (SDFAs, PDRTAs and 1-SDHAs) cannot be identified from a reasonable amount of positive data (i.e. from data whose size is polynomial in the size of the target automaton).

Significant positive results exist in learning approximately correct SDFAs from data of polynomial size. We believe that these results could be extended to the classes of PDRTAs and 1-SDHAs.

5.1 Introduction

In the learning framework of identification in the limit with probability one, the goal was to learn the exact empirical probability distribution (automaton) from a sequence of positive examples drawn according to that distribution. In the worst case, the identification takes place only in the limit, i.e. if an available sequence of examples is infinite. This learning approach could however still result in models that are good enough for certain real-world applications when a finite sequence of examples is given. Nevertheless, identification in the limit with probability one of stochastic automata remains a hard task in general.

The field of computational learning theory has offered an alternative, which, to some extent, eases the learning task. In contrast to the exact identification, the framework of *Probably Approximately Correct* learning (PAC-learning) proposed by Valiant [Val84] allows for a small divergence between the learned empirical and the target automata, limited by the threshold error ϵ. In addition, the PAC-learning criteria also restrict the probability of this error, thus permitting the divergence to be larger than ϵ, but with very small probability γ (i.e. in few cases only). The probability γ

and the error ϵ represent *confidence* and *approximation* parameters, respectively. A good overview of PAC-learning framework can be found in [KV94].

This chapter is organized as follows. First, several distance measures between stochastic automata that are relevant for PAC-learning are presented in Section 5.2. Then, the polynomial PAC-learning framework is formally defined in Section 5.3. Major positive result that SDFAs are polynomially PAC-learnable is cited and explained in this section (Theorem 12). Section 5.4 briefly presents three algorithms for learning SDFAs in the PAC setting. Finally, we conclude the chapter with prospects of polynomial PAC-learning for 1-SDHAs in Section 5.5.

5.2 Distance Measures Between Distributions

To be able to shortly present several significant positive results in PAC-learning, which could serve as a good basis for future work on PAC-learning of stochastic deterministic hybrid automata, we here show several important distance measures of divergence between two distributions L_A and $L_{A'}$ (i.e. between stochastic languages defined by automata A and A'). These measures could be used by PAC-learning algorithms in order to compare such divergence with the error parameter ϵ.

As argued in [dlH10], being able to measure the distance between distributions is important for several reasons, including:

Model selection: When several stochastic models are compared, their deviation from the target model needs to be measured.

State merging criterion in learning algorithms: It can be measured how does the divergence between models change if the states of one model are merged. The MDI algorithm (see Subsection 2.4.3) [TDdlH00] for learning stochastic deterministic finite automata uses such merging criterion.

Model classification: Stochastic models can be classified based on their similarities. These similarities can be calculated using nearest neighbor method based on a distribution distance measure.

In the following, we give an overview of several important distance measures between the stochastic automata (based on [dlH10]).

The Manhattan distance (typically denoted by d_1 or L_1) between distributions L_A and $L_{A'}$ is defined as the sum of the absolute differences between the probabilities that the strings of symbols $z \in \Sigma^*$ are drawn according to L_A and $L_{A'}$, respectively. It is calculated as follows:

$$d_1(L_A, L_{A'}) = \sum_{z \in \Sigma^*} \mid p(z|L_A) - p(z|L_{A'}) \mid . \tag{5.1}$$

The Euclidean distance (typically denoted by d_2 or L_2) between distributions L_A and $L_{A'}$ is defined as the square root of the sum of squared absolute differences between the probabilities that the strings of symbols $z \in \Sigma^*$ are drawn according to L_A and $L_{A'}$, respectively:

$$d_2(L_A, L_{A'}) = \sqrt{\sum_{z \in \Sigma^*} \mid p(z|L_A) - p(z|L_{A'}) \mid^2}. \tag{5.2}$$

Both Manhattan and Euclidean distances are special cases of d_k distance given as:

$$d_k(L_A, L_{A'}) = \sqrt[k]{\sum_{z \in \Sigma^*} \mid p(z|L_A) - p(z|L_{A'}) \mid^k}. \tag{5.3}$$

Very important distance measure for PAC-learning is the *Chebyshev distance*, which calculates the greatest difference between probabilities that the strings $z \in \Sigma^*$ are drawn according to L_A and $L_{A'}$:

$$d_\infty(L_A, L_{A'}) = \max_{z \in \Sigma^*} \mid p(z|L_A) - p(z|L_{A'}) \mid . \tag{5.4}$$

It is obtained as the limit of the expression (5.3), when $k \to \infty$. This measure expresses the absolute difference between corresponding probabilities, i.e. it works with most important (highest) values, while differences between small values are neglected.

The counterpart of the Chebyshev distance is the *logarithmic distance* that expresses only relative differences between distributions. It does not matter if these are the differences between large or small values of probabilities. Logarithmic distance is given as:

$$d_{\log}(L_A, L_{A'}) = \max_{z \in \Sigma^*} \mid \log\ p(z|L_A) - \log\ p(z|L_{A'}) \mid . \tag{5.5}$$

Another measure, commonly used by several algorithms that learn stochastic automata, is the *Kullback-Leibler* (K-L) divergence (also known as *information divergence* and *relative entropy*). In the sense of expressing the absolute or relative difference, it is somewhere between the Chebyshev and logarithmic distances. It is defined as follows:

$$d_{KL}(L_A, L_{A'}) = \sum_{z \in \Sigma^*} p(z|L_A) \log \frac{p(z|L_A)}{p(z|L_{A'})}. \tag{5.6}$$

Table 5.1 gives the classification of described distance measures with respect to the level of the importance of probabilities they consider [dlH10].

Table 5.1 Classification of distance measures. Measures on the left side express relative differences between probabilities without considering their absolute values. Measures on the right side express absolute differences.

relative difference	$\longleftrightarrow$	absolute difference
$d_{\log}$, d_{KL}, d_1, d_2, ..., d_k, d_∞		

5.3 Polynomial PAC-Learning

In this section, we recall the definition of polynomial PAC-learning, cite and explain the major result of de la Higuera, that SDFAs can be learned under this definition. This result opens prospects for polynomial PAC-learning of 1-SDHAs.

In Section 4.4, we have recalled the result of [dlHO04] that even the simplest stochastic deterministic finite automata (SDFAs) are not strong polynomially identifiable in the limit with probability one. This identification criterion is two folded: the algorithm needs to learn in time polynomial in the size of the input data, and to use data of the size polynomial in the size of the target automaton. The former has been satisfied by several SDFA learning algorithms (ALERGIA [CO94], MDI [TDdlH00]), but the latter criterion cannot be satisfied [dlHO04]. This negative result naturally generalizes to the more complex stochastic deterministic automata, such as PDRTAs and 1-SDHAs. PAC-learning framework opens space for making the polynomial approximation of the target automaton, i.e. to learn from polynomial data size while accepting certain small error.

In the following, we give definitions for an ϵ-good hypothesis (i.e. approximation of the target automaton) and for polynomial PAC-learning (due to [dlHO04] and [RST95]).

Definition 21 (ϵ-good Hypothesis). Let automata A and A' be given that belong to some class of automata C. Automaton A' is called an ϵ-good hypothesis with respect to automaton A if $d(L_A, L_{A'}) < \epsilon$ for $\epsilon \geq 0$, where $d(L_A, L_{A'})$ is some distance measure between distributions L_A and $L_{A'}$.

Definition 22 (Polynomial PAC-learning). A class of automata C is polynomially PAC-learnable using distance d, if there exists a learning algorithm Ψ and a polynomial $PolyQ$ that given any automaton $A \in C$ and any sequence of positive examples $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ of size $PolyQ(|A|)$ drawn according to the language L_A that A defines, it holds:

- with probability at least $1 - \gamma$, $\Psi(D_1, D_2, ..., D_n)$ returns an ϵ-good hypothesis with respect to automaton A, i.e.

$$p(d(L_A, L_{\Psi(D_1, D_2, ..., D_n)}) < \epsilon) \geq 1 - \gamma,$$

- $\Psi(D_1, D_2, ..., D_n)$ works in time polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\gamma}$, $|\Sigma|$, $|A|$, and $LS = \max_i(|D_i|)$, where $|D_i|$ is the length of the example $D_i \in \mathcal{D}$.

In order to present the well-known result for polynomial PAC-learning of SDFAs, the following lemma is given due to [Ang88a].

Lemma 2 ([Ang88a]). *Let L_A be any distribution on Σ^*, let $a > 0$ and let $I(n) = \sqrt{\frac{6a(logn)}{n}}$. Then:*

- *with probability at least $1 - n^{-a}$,*
- *with probability one and for all but a finite number of values of n:*

$$d_\infty(L_A, L_{A'}) \leq I(n), \tag{5.7}$$

where $L_{A'}$ is the empirical distribution built from an input data of size n.

Theorem 12 ([dlHO04]). *The class of SDFAs is polynomially PAC-learnable using Chebyshev distance measure d_∞.*

Proof. (Sketch, [dlH10], [dlHO04]) Even the simplest algorithm Ψ that only builds a prefix tree acceptor whose size is polynomially linked with the size of the learning data $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ satisfies the criteria from Definition 22, when the

Chebyshev distance $d_\infty(L_A, L_{\Psi(D_1,D_2,\ldots,D_n)})$ is used. The maximum value of this distance is one by definition and moreover it is bounded by the value $I(n)$ from the expression (5.7), which converges very fast as n grows. □

Observation 6. *The result from Theorem 12 no longer holds when distance measures $d_1(L_A, L_{\Psi(D_1,D_2,\ldots,D_n)})$ and $d_2(L_A, L_{\Psi(D_1,D_2,\ldots,D_n)})$ are used. The reason is that these measures are unbounded and can in general case grow with n.*

In order to illustrate convergence of the bound $I(n)$ from the expression (5.7), the following example is given.

Example 2. *Table 5.2 shows the influence of the number n of compared examples of two distributions L_A and $L_{A'}$ on distribution distance measures $d_1(L_A, L_{A'})$ and $d_2(L_A, L_{A'})$, and on the bound $I(n)$ given in Lemma 2 (parameter $a = 2$). Both sets of examples are generated from the standard uniform distribution on the open interval $(0, 1)$. Due to their cumulative property, both distance measures grow with n. On the other hand, the value $I(n)$ that bounds the distance $d_\infty(L_A, L_{A'})$ decreases with n.*

Table 5.2 The influence of the number of examples on several distance measures.

n	10	10^2	10^3	10^4	10^5	10^6
$d_1(L_A, L_{A'})$	5.1682	31.4561	321.7636	$3.2813 \cdot 10^3$	$3.3355 \cdot 10^4$	$3.3338 \cdot 10^5$
$d_2(L_A, L_{A'})$	1.8779	3.9528	12.5206	40.2572	129.0636	408.4201
$I(n)$	1.6623	0.7434	0.2879	0.1051	0.0372	0.0129

The influence of the number of examples n on the bound $I(n)$ is also shown graphically in Figure 5.1. In addition, the figure shows the probability that the distance $d_\infty(L_A, L_{A'})$ is bounded by $I(n)$. It can be seen that in this example the probability of the expression (5.7) converges to one very fast.

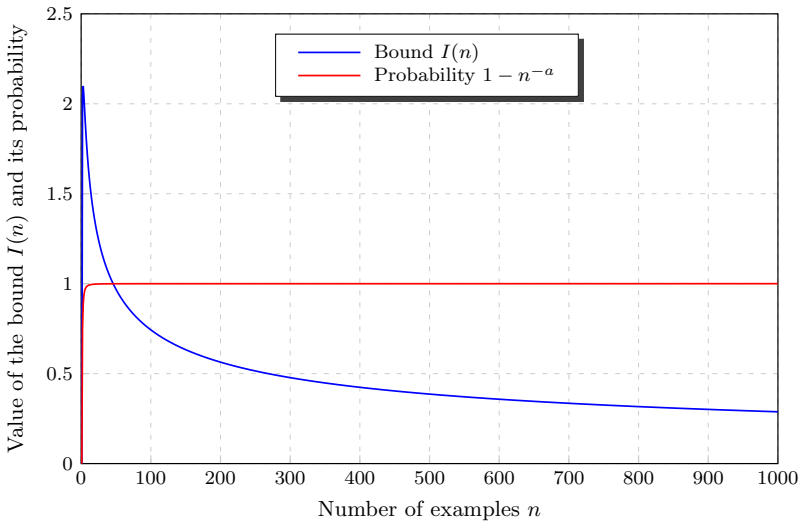


Fig. 5.1 Influence of n on bound $I(n)$ and the probability $1 - n^{-a}$.

When learning automata in the PAC setting, it is worth to note the significant work, which has been done in modeling noise. This work has been motivated by the fact that every information source is in practice susceptible to noise. Noise, which can occur both in the learning examples themselves, as well as in their class labels, can have big effect on learnability. Although in learning stochastic deterministic automata from text only positive examples exist, still (like in the presence of both positive and negative examples) some external instance (oracle, domain expert) needs to provide the information if the examples are positive. In other words, learning from text can also be influenced by the class noise. Several specific noise models have been developed over years such as *constant partition classification noise* [Dec97], *random classification noise* [AL88], and *malicious classification noise* [Slo95]. These noise models are uniformly studied under the statistical query model [Kea98], but as argued in [JHZ12], this model is hardly extendable to new noise models and it does not permit the study on noise properties, which would enable PAC-learning from noisy data. This problem was addressed in the same paper that introduced a novel framework for a wide class of noise models, under which PAC-learning can be guaranteed. Such an approach could be easily extended to guarantee learning in the limit from both informant and text.

5.4 Algorithms for Polynomial PAC-Learning of SDFAs

Here we briefly present three existing algorithms for polynomial PAC-learning of SDFAs.

Early results on polynomial PAC-learning of stochastic deterministic finite automata (SDFAs) were negative. In [KMR$^+$94] a relation has been established between certain parity functions and distributions generated by 2-letter SDFAs. Under generally accepted assumption that such parity functions are not polynomially PAC-learnable, it has been concluded that this result also holds for the whole class of SDFAs.

However, several algorithms have been developed that use rigorous constraints in order to make the learning of SDFAs in the PAC learning framework possible. They are presented in the following.

Learn-Acyclic-PFA: The pioneer algorithm was given by Ron et al. [RST95] and called *Learn-Acyclic-PFA*. It polynomially PAC-learns acyclic distinguishable SDFAs. The definition of distinguishability is given as follows (due to [RST95]):

Definition 23 (Distinguishability). For $0 \leq \mu \leq 1$ two states $s_1, s_2 \in S$ of the automaton A are called $\mu - distinguishable$ if there exists a string $z \in \Sigma^*$ for which $|p(s_1, z) - p(s_2, z)| \geq \mu$, where $p(s_1, z)$ and $p(s_2, z)$ are probabilities that the automaton will generate a string that starts in s_1 and s_2, respectively. Automaton A is $\mu - distinguishable$ if its every pair of states is $\mu - distinguishable$.

$\mu - distinguishable$ SDFAs are easier to learn because it is guaranteed that for any 2 states a string exists whose difference in probability when using these states as its initial states is above the given threshold μ [dlH05]. The distance measure used by *Learn-Acyclic-PFA* for calculating the error between the target and the hypothesis automata is K-L divergence, given by expression (5.6). Although this algorithm works with distributions over Σ^*, it is basically restricted to the

automata of bounded size (i.e. Σ^k for some fixed k), as it learns only acyclic SDFAs. It has been tested in two applications: the cursive handwriting recognition system and learning pronunciation models for spoken words. Models have shown good performance, especially in comparison to hidden Markov models and their large learning times [RST95].

C-T Algorithm: The first algorithm that could polynomially PAC-learn the whole class of SDFAs (both acyclic and cyclic) has been proposed by Clark and Thollard in [CT04]. This algorithm is sometimes denoted as $C-T$ algorithm [CG08]. The used distance measure between the target and hypothesis automata is again the K-L divergence. Therefore, it is said that it KL-PAC learns SDFAs. In contrast to *Learn-Acyclic-PFA*, the $C-T$ algorithm can learn all distributions (languages) over Σ^*. In addition, yet another parameter is introduced, namely an upper bound on the expected length of generated strings. The parameters that the $C-T$ algorithm requires as input are the alphabet size $|\Sigma|$, confidence γ, approximation ϵ, distinguishability μ, and the bound on the longest string LS that can be generated from any automaton state. Moreover, for successful polynomial PAC-learning, the algorithm assumes that the size of the target automaton is at most $|A|$ states and that A is $\mu - distinguishable$. Then the algorithm computes the number $n = poly(|\Sigma|, |A|, ln(\frac{1}{\gamma}), \frac{1}{\epsilon}, \frac{1}{\mu}, LS)$ and requests the input data $\mathcal{D}$ of the size n. Only then, it applies fairly standard state merging technique and processes the data $\mathcal{D}$ again using the parameters $|A|, \epsilon, \mu$, and LS while merging. Eventually it polynomially KL-PAC learns the target automaton.

The algorithm of Castro and Gavaldà: As argued in [CG08], although the $C-T$ algorithm is indeed a polynomial PAC-learning algorithm, it has several practical drawbacks. First, during the computation it uses a number of parameters including the distinguishability parameter μ. Its value is unknown and has to be guessed, using case-based *trial-and-error* procedure. This is in most practical cases a hard task. Second issue is that although the $C-T$ algorithm uses the input data of the polynomial size $n = poly(|\Sigma|, |A|, ln(\frac{1}{\gamma}), \frac{1}{\epsilon}, \frac{1}{\mu}, LS)$, the order of the polynomial easily gets overwhelming (in [CG08] the order of 10^{24} is obtained for fairly low values of parameters: $|\Sigma| = 2$, $|A| = LS = 6$, and $\epsilon = \gamma = \mu = 0.1$). This happens because it always computes and requires the worst case data size. In order to overcome these issues, Castro and Gavaldà proposed the improved $C-T$ algorithm [CG08]. It does not need any information about the distinguishability of the target automaton as the input. Moreover, the input basically consists of only parameters $|\Sigma|$ and γ. It takes the available data $\mathcal{D}$ and tries to extract as much information as possible from them, which is from the practical point of view much more realistic setting. During computation, it uses no parameters whatsoever, which is its biggest advantage. The approach has been demonstrated in learning few small targets (8 to 10 states), as well as using the real-world dataset coming from an ecommerce site (travel agency, nontrivial structures with 30-50 states).

5.5 Prospects of Polynomial PAC-Learning for Hybrid Automata

Looking at Theorem 12 and several mentioned polynomial PAC-learning algorithms for SDFA, the question naturally arises if one-clock stochastic deterministic hybrid automata could be learned under the PAC-learning setting. Based on the fact that

just like SDFAs, probabilistic deterministic real-time automata can also be identified in the limit with probability one, Verwer argues in [Ver10] that they could probably also be polynomially PAC-learnable. We believe that this could also be the case for 1-SDHAs. Our algorithm HyBUTLA (given in Chapter 6), which learns 1-SDHAs in the limit with probability one from text, needs to be adjusted to measure the error in merging the states. In addition, a polynomial would have to be found that limits the amount of the learning examples that is necessary to bound this error. However, this work remains to be done in future.

Part III

Algorithms

Chapter 6

Automated Learning of 1-SDHAs from Data

In this chapter, we present our major contribution: the approach for automated learning of 1-SDHAs from data. The main results given here are as follows:

- we describe how a set of symbols, which trigger transitions between automaton states, and those transitions' timing constraints, are generated from logged data,
- we give the HyBUTLA algorithm that learns 1-SDHA models for hybrid production systems,
- we prove that the HyBUTLA algorithm identifies the class of 1-SDHAs in the limit with probability one from positive learning examples only,
- we show that our algorithm runs in time polynomial in the size of the input data,
- the expert knowledge needed for a successful application of the HyBUTLA algorithm is identified.

The key property that makes the class of 1-SDHAs identifiable in the limit with probability one by the HyBUTLA algorithm from positive examples only, is that the examples come from a well-defined probability distribution. Statistical regularity in such data is able to compensate for the lack of negative examples [Ang88a].

The structure of this chapter is as follows. Section 6.1 describes the means of data acquisition from distributed production systems. In addition, the existing technology for filtering noisy data in industrial facilities is explained. The data we used for learning in Section 8.2 of this thesis have been filtered using this technology. We illustrate how we generate a set of symbols and a set of transition timing constraints of 1-SDHA from logged measurements in Section 6.2. Section 6.3 finally presents our HyBUTLA algorithm in several steps. The means of detecting abrupt changes in system continuous dynamics is given in Section 6.4. Special attention is devoted to the operation of splitting an automaton state in Section 6.5. This operation is based on the abrupt change detection and makes it easier and faster to approximate the continuous dynamics of a modeled system. In Section 6.6, we prove the convergence and polynomial runtime properties of the HyBUTLA algorithm (Theorem 14). The conclusion is given in Section 6.7, together with the required expert knowledge for our learning approach. This chapter is based on materials that we partially published in [NMVJ11, VKBNM11a, VKBNM11b, NSV+12, FFP+12, Vod12, Vod13].

6.1 Data Acquisition and Preprocessing

Before any learning from data could take place, data acquisition and preprocessing need to be performed in a real-world system. In the following two subsections, we give a very brief introduction to several basic techniques for acquiring and preparing the data for modeling tasks. Readers interested in more details about these techniques can follow the corresponding given references.

6.1.1 Generic Data Acquisition for Distributed Production Systems

The main prerequisite of our approach for automated learning of behavior models is having the logs of relevant discrete and continuous signals of a system. Acquisition of such process data is in complex (distributed) hybrid production systems a challenging task. As we emphasized in [FFP+12], this is due to the following issues:

Heterogeneous automation systems: A large variability of communication protocols, interfaces, machinery, and control devices from different vendors can be found in a typical production system. Obtaining unified data from several system modules that comprise such different elements is not easy.

Timing requirements and synchronized measurements: Different modules of an automation system (e.g. sensors) often do not use the same time base. Therefore, process data originating at these modules need to be synchronized in time with an acceptable accuracy.

Data integration: Existing solutions for data acquisition often use proprietary interfaces. They prevent the data to be accessed in an easy way.

These obstacles are illustrated in Figure 6.1. It shows an example of a shop floor with two production cells, each containing two production modules. Different industrial network protocols, such as Modbus/TCP, PROFINET or EtherCAT, can be found in these systems.

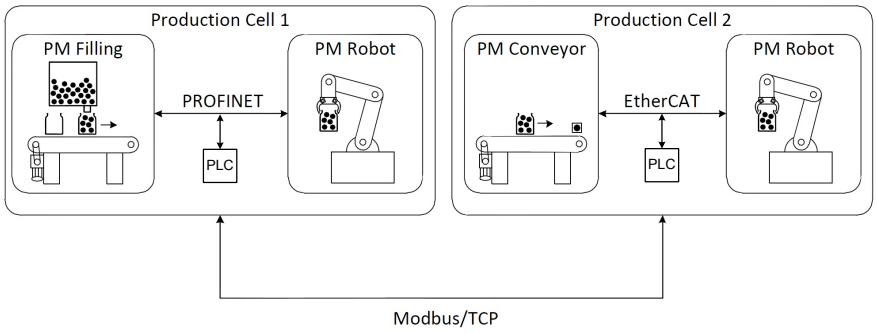


Fig. 6.1 An example of a shop floor communication (reproduced from [PKN+12] with added *Programmable Logic Controllers* (PLCs)).

Solutions to aforementioned issues are given in [PN12] and further extended with application examples in [PKN+12]. Since these solutions were applied in acquisition

of data used for learning models in one part this work (Section 8.2), we summarize them here without entering unnecessary details. More information can be found in the corresponding papers.

The architecture of generic synchronized data acquisition approach is given in Figure 6.2. The main idea is to place distributed probes, called *dataloggers*, in all relevant segments of industrial networks. They log communication between drives, actuators, sensors, and *Programmable Logic Controllers* (PLCs) without influencing the normal network traffic. As they can be easily adapted to different network protocols such as PROFIBUS [PM08] or EtherCAT [Eth08], they are suitable for data acquisition in heterogeneous automation systems. The issue of time synchronization is solved using the *precision time protocol* defined in the IEEE 1588 standard [IEC04]. Finally, data integration is enabled through the *OPC Unified Architecture* (OPC-UA) [MLD09]. OPC-UA server receives the data from dataloggers via UDP/IP interface, integrates them, and makes them accessible using an uniform interface and standardized protocols such as SOAP/http (for Windows clients or smart phone platforms) or OPC-UA binary protocol (for *Supervisory Control And Data Acquisition* (SCADA) industrial control systems [Boy08]).

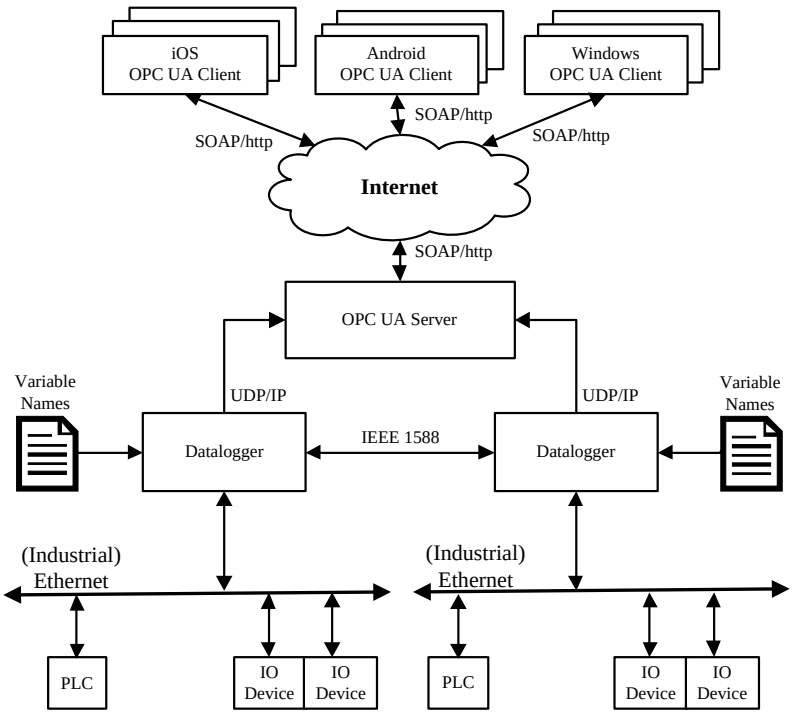


Fig. 6.2 The architecture of generic synchronized data acquisition approach (reproduced from [PN12]).

6.1.2 Dealing with Measurement Noise in Industry

Our learning algorithm learns models using data that are logged during the runtime of a modeled system. As mentioned before, these logs include both control signals and measurements of continuous process (physical) variables. In production facilities, such measurements are always prone to corruption due to several errors. Sources of measurement errors could be, for example, poor cabling practices, uncalibrated instruments, environmental changes, or induced measurement noise, which occurs when a measurement is transmitted from one point to another point in a system [HC01a]. Clever handling of measuring devices could prevent such errors to a large extent, but typically it is not possible to completely remove signal noise. Therefore, advanced filtering and other signal processing technologies are applied before measurements are logged in a database. Filtering is a procedure for decreasing the measurement noise (called *denoising*), i.e. filters ideally enable propagation of only desired frequency ranges of a signal.

Before we explain where the filtering takes place in an industrial facility, it is important to note the following. Physical signals (process variables such as pressures, temperatures etc.), are in the industry typically measured, converted to an electric current signal and scaled to the range of 4-20 mA. Conversion elements used to this aim are called *transducers*. Transducers that incorporate processing of the measured signal are typically called *transmitters* [HC01a]. After conversion to the electrical signal, a measurement is transmitted through the industrial network back to control system, where it is used for controlling purposes and logged in a database. Before logging, it is converted back to its original physical unit. An example of a program for converting and scaling a physical variable in an industrial context is the Siemens function FC41 [Man06].

Based on the size and complexity of an industrial facility, filters for denoising can be implemented in its different parts. In the case of smaller plants that use simpler control systems, sufficient filtering typically takes place already in transmitters. They comprise several elements for measuring, converting, scaling, and denoising physical variables. An example of transmitters for pressure, differential pressure, flanged level, and absolute pressure are Siemens SITRANS P Series DSIII Transmitters [Man10]. Complex and large plants are controlled by distributed control systems such as Siemens TELEPERM XP, which incorporate more advanced denoising filters in the control system itself [Man94]. In such plants, denoising is often done both in transmitters and in control system. Logged data that we use for learning models are typically already denoised, using one of these approaches.

6.2 Generating Alphabet and Timing Constraints from Measurements

In this section, we illustrate how we generated a set of symbols, which trigger transitions between the automaton states, and those transitions' timing constraints (an alphabet Σ and a set Δ according to Definition 13) from logged measurements of a running system. These measurements come in a form of a sequence of learning examples that is defined in Section 3.4.

According to Definition 9, each learning example D_i is a matrix of values:

$$D_i = \begin{bmatrix} t_1^i & q_{1,1}^i & \cdots & q_{d,1}^i & u_{1,1}^i & \cdots & u_{c,1}^i & y_{1,1}^i & \cdots & y_{o,1}^i \\ t_2^i & q_{1,2}^i & \cdots & q_{d,2}^i & u_{1,2}^i & \cdots & u_{c,2}^i & y_{1,2}^i & \cdots & y_{o,2}^i \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ t_l^i & q_{1,l}^i & \cdots & q_{d,l}^i & u_{1,l}^i & \cdots & u_{c,l}^i & y_{1,l}^i & \cdots & y_{o,l}^i \end{bmatrix},$$

where $j = 1, ..., l$, $t_j^i \in \mathbb{N}$ are the time stamps, $k = 1, ..., d$ and $j = 1, ..., l$, $q_{k,j}^i \in \{0, 1\}$ are the values of discrete binary signals, $p = 1, ..., c$ and $j = 1, ..., l$, $u_{p,j}^i \in \mathbb{R}$ are the values of continuous input signals, and $r = 1, ..., o$ and $j = 1, ..., l$, $y_{r,j}^i \in \mathbb{R}$ are the values of continuous output signals. In the following example, we show how an alphabet Σ and a set of transition timing constraints Δ are generated from such matrices.

Example 3. *Let a learning example D_i be recorded in a hybrid system with three discrete control signals and three continuous process variables. Let it be given as the following matrix with 10 rows (i.e. logged samples from a system):*

$$D_i = \begin{bmatrix} 1 & \boxed{0\ 0\ 0} & 1.3 & 9.6 & 14.5 \\ 2 & 0\ 0\ 0 & 1.5 & 9.5 & 14.4 \\ 3 & \boxed{0\ 0\ 1} & 1.8 & 9.3 & 14.1 \\ 4 & 0\ 0\ 1 & 2.1 & 8.9 & 13.6 \\ 5 & 0\ 0\ 1 & 2.2 & 8.5 & 13.3 \\ 6 & \boxed{0\ 1\ 1} & 2.3 & 8.4 & 13.2 \\ 7 & 0\ 1\ 1 & 2.4 & 8.2 & 13.1 \\ 8 & 0\ 1\ 1 & 2.6 & 5.1 & 12.9 \\ 9 & \boxed{0\ 0\ 1} & 2.9 & 7.9 & 12.7 \\ 10 & 0\ 0\ 1 & 3.1 & 7.8 & 12.6 \end{bmatrix}.$$

Initially, symbols of an alphabet Σ are defined as changes in discrete signals only. Discrete parts of samples in which these changes take place are marked in the matrix above. They represent the symbols of an alphabet. The timing of each such sample relative to the timing of the previous such sample defines a corresponding symbol's timing constraint. Therefore, the given learning example defines the following symbols and timing constraints:

$$\begin{aligned} a &= 000, \delta_1 = [1, 1] \\ b &= 001, \delta_2 = [3 - 1, 3 - 1] = [2, 2] \\ c &= 011, \delta_3 = [6 - 3, 6 - 3] = [3, 3] \\ b &= 001, \delta_4 = [9 - 6, 9 - 6] = [3, 3], \end{aligned}$$

where $a, b, c \in \Sigma$ and $\delta_1, \delta_2, \delta_3, \delta_4 \in \Delta$. Of course, the first symbol of the alphabet is always defined by the discrete part of the example's first sample. Consequently, this symbol's timing constraint is always $\delta_1 = [1, 1]$.

A set of symbols Σ and a set of transition timing constraints Δ, generated as explained in this example, represent an input to our HyBUTLA learning algorithm. We present it in the following section.

6.3 The HyBUTLA Learning Algorithm

In this section, we present our *Hybrid Bottom-Up Timing Learning Algorithm* (HyBUTLA) that learns behavior models for components of a hybrid system in the formalism of one-clock stochastic deterministic hybrid automata, which use the relative timing (i.e. only the time between consecutive events is modeled). This algorithm identifies the correct automaton structure (states and transitions), transition timing constraints, transition and ending state probabilities, as well as functions that approximate the continuous dynamics of a system. Its main characteristics are as follows:

Modeling continuous behavior: By learning functions that approximate the continuous dynamics, the HyBUTLA algorithm models the continuous behavior of a system. Each automaton state, created based on a change in discrete control system, is associated with a certain portion of logs of continuous input and output signals (process variables). These logs, together with their timing information, are used as input for machine learning methods. In that way the functions that approximate the continuous dynamics are learned. Depending on the application, a number of such methods can be used with the HyBUTLA algorithm, ranging from linear regression for piecewise linear systems, to neural networks for highly nonlinear systems.

Bottom-up merging order: The HyBUTLA algorithm is, together with its related algorithm BUTLA for learning timed automata [MNV+11], the first algorithm that uses the bottom-up merging order. This improves the algorithm runtime [NSV+12] as the recursive compatibility checks of large subtrees are avoided.

Novel learning of time: For each symbol of the given alphabet (corresponding to a discrete change in a system[1]), a *Probability Density Function* (PDF) of its timing is computed over all available learning examples. By finding minima in that PDF, it is established if it represents the sum of several Gaussian distributions. When this is the case, those changes in the system are treated separately despite the same symbol they have. This has a huge impact on the learning process, because transitions that are triggered by the same symbol cannot be merged due to different Gaussian distributions whom their timings belong to. The same procedure is applied in the BUTLA algorithm [MNV+11] (please recall Figure 2.15 in Subsection 2.4.3).

Besides these main characteristics, let us emphasize once more that the HyBUTLA algorithm is an offline algorithm that learns 1-SDHAs in the limit with probability one from text (i.e. from positive learning examples only).

The HyBUTLA algorithm identifies 1-SDHAs using the approach illustrated in Figure 6.3. The pseudocode of the algorithm is given in Figure 6.4. It uses the general state merging approach to learning (recall Figure 2.13 in Subsection 2.4.1). The steps of the learning approach, together with corresponding lines of the pseudocode, are described in the following.

Step 0: Sensor readings of all relevant discrete signals and process variables are logged in a database over multiple production cycles of a hybrid system. These readings include the timing information. In general, the more logged cycles, the

[1] In general, symbols can be assigned to other types of changes in a system, i.e. changes in continuous signals or valuations of clocks (timers).

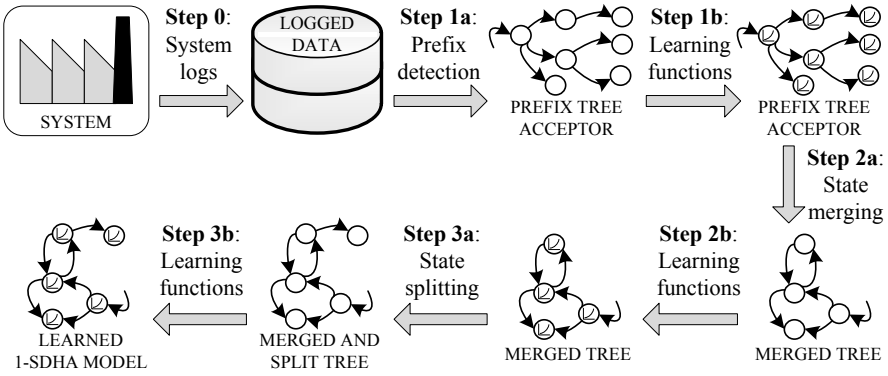


Fig. 6.3 The approach for learning 1-SDHAs with the HyBUTLA algorithm.

Given:
(1) Symbols Σ with their timings Δ based on changes in discrete signals $q \in Q$
(2) Sequence of positive examples $\mathcal{D}$
Result: 1-SDHA A

(0) $\forall a \in \Sigma$ find sum of PDFs: $PDF(a) := \sum_{\tau=(*,a,*,\delta)\in T} \delta$, where $*$ is an arbitrary element. By finding minima in $PDF(a)$, segment it in clusters $d_i(a), i \in \mathbb{N}$.
(1) Build prefix tree $A = (S, s_0, \Sigma, T, \Delta, P, x, \Theta)$ based on $\mathcal{D}$. A is a 1-SDHA according to Definition 14.
(2) $\forall s \in S$ learn $\theta_s \in \Theta$ using the time t, inputs $u \in U$, and outputs $y \in Y$
(3) **for all** $s, s' \in S$ in a bottom-up order
(4) **if** *compatible*(s, s') **then**
(5) A = *merge*(s, s')
(6) *determinize*(A)
(7) **end if**
(8) **end for**
(9) A = *split*(A)
(10) **return** A

Fig. 6.4 The HyBUTLA algorithm.

better. The logs of production cycles represent a sequence of learning examples $\mathcal{D}$, which is given to the HyBUTLA algorithm as an input. Since we are interested in learning normal behavior, only logs taken under normal operating conditions of a system are used. Therefore, a sequence $\mathcal{D}$ is a sequence of positive learning examples. Controlled jumps (changes in discrete signals Q) in all examples $D_i \in \mathcal{D}$ are used to define a finite set of symbols Σ, as explained in Section 6.2. The timing of the occurrence of each symbol $a \in \Sigma$ relative to the timing of the occurrence of its preceding symbol is used for defining its timing interval (constraint) $\delta \in \Delta$. The sets Σ and Δ also represent inputs to the HyBUTLA algorithm. Timing constraints δ, expressed as *Probability Density Functions* (PDFs), are combined for every symbol $a \in \Sigma$ (see Figure 2.15 in Subsection 2.4.3). By finding minima of their combined PDFs, the time clusters $d_i(a), i \in \mathbb{N}$ are formed (line 0 in Figure 6.4). Later on, during the merging process, these clusters will serve to check if timings of two transitions are similar enough, i.e. if they belong the same time cluster.

Step 1: After the creation of time clusters, the examples are ordered lexicographically, i.e. sorted by length and the alphabet[2]. If the timings of common symbols from different learning examples belong to the same time cluster, they are combined (Step 1a in Figure 6.3, line 1 of the HyBUTLA algorithm) in a tree-like structure called the *Prefix Tree Acceptor* (PTA). Every example represents a path in a PTA (i.e. a sequence of states and transitions), which starts at the initial state. Different examples can share parts of the paths. For a better understanding of this process, we have illustrated it in Figure 6.5. An exemplary system has three discrete binary signals, namely q_1, q_2, and q_3. Controlled jumps (changes) of these signals are shown for three learning examples: D_1, D_2, and D_3. It can be seen that every specific controlled jump corresponds to a specific symbol of the alphabet Σ in the following way:

- "OPEN" event of the signal q_1 triggers the symbol a,
- "OPEN" event of q_2 triggers b,
- "OPEN" event of q_3 triggers f,
- "CLOSE" event of q_1 triggers c,
- "CLOSE" event of q_2 triggers e.

The figure also shows the PTA constructed from these examples. When symbols a, b, and c in all three examples D_1, D_2, and D_3 have the timing intervals that belong to the same time cluster, they are combined in one branch of the PTA. Its timing intervals are extended to include the minimum and the maximum timing of their corresponding transitions. The symbol f is specific for example D_1 only, and therefore the transition it triggers is not combined with any other in the PTA. In contrast, symbol e exists in both D_2 and D_3, but since its timings in these two examples belong to different time clusters, they are not combined in one branch. After a structure of the PTA has been created, continuous data corresponding to logs of continuous input signals $u \in U$, continuous output signals $y \in Y$, and the time t are used for learning functions $\theta_s \in \Theta, \forall s \in S$ (Step 1b in Figure 6.3, line 2 of the HyBUTLA algorithm). For each state s, only portions of continuous data that were logged when the system was in that particular state are used for learning its θ_s function. Various learning methods can be used, ranging from relatively simple multiple linear regression with linear terms, to much more complex support vector regression or neural networks. The approximation accuracy is expressed as the coefficient of determination $R^2(\theta)$ [MC04], which is a value between 0 and 1 that shows the portion of variability in the observed data y_j that is accounted by the learned θ function. It is defined as:

$$R^2(\theta) = 1 - \frac{SS_{error}}{SS_{total}} = 1 - \frac{\sum\limits_j (y_j - \theta_j)^2}{\sum\limits_j (y_j - \overline{y})^2}, \tag{6.1}$$

where SS_{error} is the sum of squares of errors, SS_{total} is the total sum of squares, θ_j are values of the learned θ function at j points, and $\overline{y}$ is the mean of observed data y_j. Basically, $R^2(\theta)$ is a measure of the goodness of fit that describes how well a learned θ function fits y_j data points. An $R^2(\theta)$ value closer to 0 indicates a very poor fit, while value closer to 1 indicates a good fit.

[2] This sort is typical for some algorithms (e.g. ALERGIA) and it enables easier enumeration of automaton states.

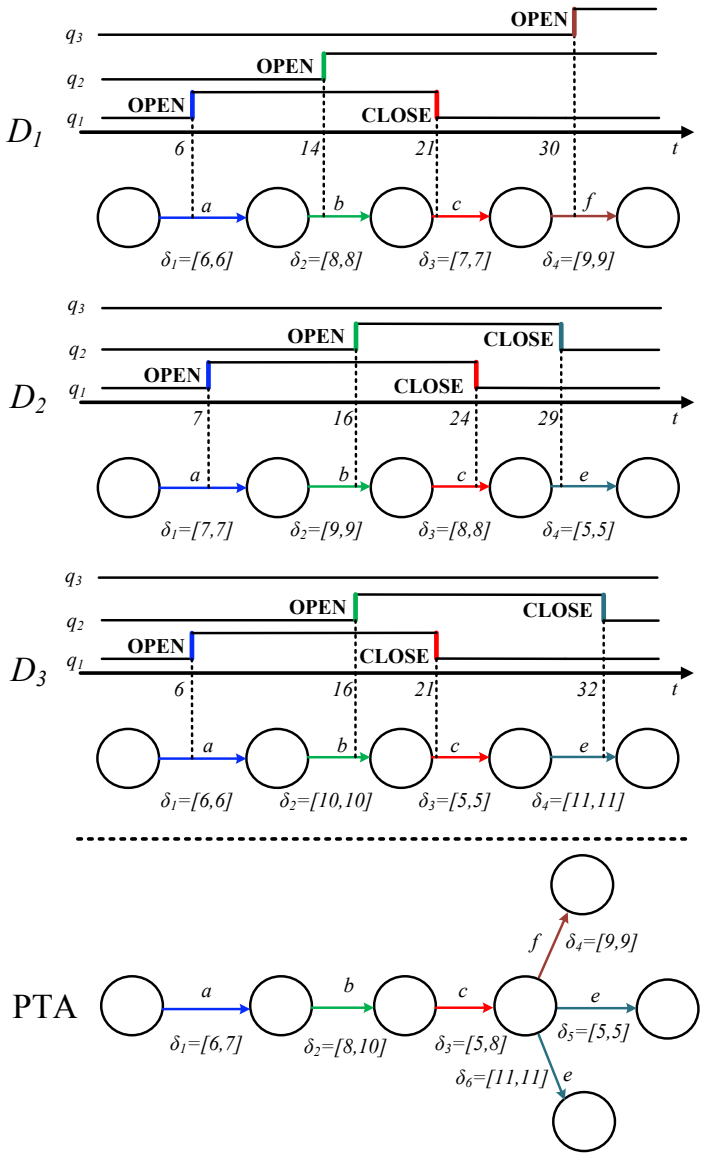


Fig. 6.5 An illustration of building a prefix tree acceptor from learning examples.

Step 2: A PTA created in the previous step is just a smart way of representing the learning data. It describes an informational content of only those examples that were used for learning. Thus, it hardly generalizes to unseen examples. Typically, a PTA has a large number of states even for small systems with only a few signals and short production cycles. Therefore, it is not easily visualized and understood by human experts. For all these reasons, PTA states are merged (Step 2a in Figure 6.3). First, the compatibility is checked for all states in a bottom-up order. States found to be compatible with respect to the criteria given below are merged. Their portions of continuous data used for learning θ functions are merged as well, and a new θ function is learned for the newly created state (Step 2b in Figure

6.3). Since merging can produce non-determinism, (i.e. several transitions which are triggered by the same symbol and whose time intervals belong to the same time cluster could appear from one source state to several destination states) a *determinization* procedure is performed. It merges non-deterministic outgoing transitions and their corresponding states recursively. Step 2 is implemented in lines 3-8 of the HyBUTLA algorithm in Figure 6.4.
The function *compatible* for testing the compatibility between two states s and s' is given in Figure 6.6. It checks whether the probabilities for taking a specific transitions or for ending in the state are similar enough for these states. We use the same statistical measure of similarity that was used for learning SDFAs in the ALERGIA algorithm [CO94]. This measure is based on the Hoeffding bound [Hoe63], which shows that given a Bernoulli variable of probability p and observed frequency of $\frac{f}{g}$, then with probability of at least $1-\alpha, \alpha \in \mathbb{R}, \alpha > 0$ it holds:

$$\left| p - \frac{f}{g} \right| < \sqrt{\frac{1}{2g} \log\left(\frac{2}{\alpha}\right)}. \tag{6.2}$$

We implemented the similarity measure of the ALERGIA algorithm [CO94] as the function *fractions-different*, which tests whether two fractions $\frac{f_0}{g_0}$ and $\frac{f_1}{g_1}$ are significantly different:

$$\begin{aligned} &\textit{fractions-different}(g_0, f_0, g_1, f_1) := \left| \frac{f_0}{g_0} - \frac{f_1}{g_1} \right| > \\ &\sqrt{\frac{1}{2} \log\left(\frac{2}{\alpha}\right)} \left(\frac{1}{\sqrt{g_o}} + \frac{1}{\sqrt{g_1}} \right), f_0, g_0, f_1, g_1 \in \mathbb{N}. \end{aligned} \tag{6.3}$$

Since it involves a pair of Bernoulli variables, the probability that this function will return the correct answer is at least $(1-\alpha)^2, \alpha \in \mathbb{R}, \alpha > 0$ [CO99]. When the inequality is true, two fractions are considered to be too different. First, the following quantities are calculated in the function $compatible$ (Figure 6.6):

- the number of examples that attain a state s with a transition triggered by a symbol a whose timing constraint is δ ($\forall a \in \Sigma, \forall \delta \in \Delta$ and $\forall s \in S$, line 1),
- the number of all examples that enter a state $s', \forall s' \in S$ (line 2),
- the number of all examples that leave a state $s, \forall s \in S$ (line 3),
- the number of all examples that end in a state $s, \forall s \in S$ (line 4).

These quantities are obtained by simply counting the number of examples that trigger the corresponding transition $\tau \in T$. The examples that end in a state are also counted for estimating the ending state probability. For these counts, a simple function: $Num : T \to \mathbb{N}$ is used.
The similarity of ending state probabilities is computed for two states s and s', using the expression (6.3). If these probabilities are too different, the function $compatible$ returns *false* and the states are declared as not compatible (lines 5-7 in Figure 6.6). Otherwise, it is checked for each specific symbol $a \in \Sigma$ if the probabilities of the incoming transitions are similar (lines 8-9 in Figure 6.6). In addition, the membership of those transitions' timing intervals $\delta \in \Delta$ to the same

Given: $s, s' \in S$
Result: decision true or false
(1) $f(a, \delta, s) := \sum_{\tau=(*,a,\delta,s)\in T} Num(\tau), a \in \Sigma, \delta \in \Delta, s \in S$ where $*$ is an arbitrary element.
(2) $f_{in}(s') := \sum_{\tau=(*,*,s',*)\in T} Num(\tau), s' \in S$
(3) $f_{out}(s) := \sum_{\tau=(s,*,*,*)\in T} Num(\tau), s \in S$
(4) $f_{end}(s) := f_{in}(s) - f_{out}(s), s \in S$
(5) **if** *fractions-different*$(f_{in}(s), f_{end}(s), f_{in}(s'), f_{end}(s'))$ **then**
(6) **return false**
(7) **end if**
(8) **for all** $a \in \Sigma$
(9) **if** *fractions-different*$(f_{in}(s), f(a, \delta, s), f_{in}(s'), f(a, \delta', s))$ **then**
(10) **return false**
(11) **end if**
(12) **if** $\delta \in d_i(a)$ **and** $\delta' \in d_j(a)$ **and** $i \neq j$ **then**
(13) **return false**
(14) **end if**
(15) **if** not *compatible*(s_1, s_2) where $(s, a, s_1, \delta_1), (s', a, s_2, \delta_2) \in T$ **then**
(16) **return false**
(17) **end if**
(18) **end for**
(19) **return true**

Fig. 6.6 The function *compatible.*

time cluster is tested (line 12 in Figure 6.6). When both conditions are satisfied for states s and s', the compatibility of their corresponding subtrees is checked too. This check is performed by calling the function $compatible$ recursively for all states in subtrees (line 15 in Figure 6.6). Finally, when all criteria are satisfied, the states are declared as compatible and can be merged as well as their subtrees. Estimations of transition and ending state probabilities are made using the function Num in the following way. The number of examples that attain a state $s \in S$ equals the sum of the number of examples that end in a state and the number of examples that leave the state:

$$\sum_{(\tau=(s'',b,s,\delta')\in T), s''\in S, b\in\Sigma, \delta'\in\Delta} Num(\tau) =$$
$$Num(s, \lambda, s, \delta_0) + \sum_{(\tau'=(s,c,s',\delta)\in T), s'\in S, c\in\Sigma, \delta\in\Delta} Num(\tau'),$$

where λ is an empty string. It does not trigger any transitions and we denote its timing constraint as $\delta_0 = [0, 0]$. Now we can get the expression for calculating the probability of a transition (s, a, s', δ):

$$p(s, a, s', \delta) = \frac{Num(s, a, s', \delta)}{\sum_{(\tau=(s'',b,s,\delta')\in T), s''\in S, b\in\Sigma, \delta'\in\Delta} Num(\tau)}. \tag{6.4}$$

The probability that the example ends in a state s is then as follows:

$$p(s, \lambda, s, \delta_0) = 1 - \sum_{a\in\Sigma, s'\in S, \delta\in\Delta} p(s, a, s', \delta). \tag{6.5}$$

Please note that in addition to testing the similarity of ending state and transition probabilities, as well as the membership of those transitions' time intervals to

the same time cluster, the initial version of our HyBUTLA algorithm [NMVJ11, VKBNM11a, VKBNM11b] included similarities of θ functions of two states candidates for merging in the compatibility criteria. This additional compatibility check highly depended on the type of regression functions used for approximation. When multiple linear regression was used, it was checked if at least 50% of the corresponding regression coefficients are similar, i.e. if their relative difference is not larger than some predefined threshold (e.g. 5% or 10%). When neural networks are used for regression, the similarity of states' θ functions had to be defined differently, e.g. as similarity of the corresponding neurons' biases and weights. This compatibility criterion introduced an extensive complexity to our algorithm, and was therefore removed from the version of the algorithm that was used later on [NSV$^+$12, Vod12, FFP$^+$12, Vod13, VMN13].

Step 3: In order to speed up the algorithm, fast machine learning methods for learning Θ functions can be used. However, they may suffer from the lack of accuracy. For that reason, the additional splitting step is provided (Step 3a in Figure 6.3, line 9 in Figure 6.4). In general, it finds those states with small function approximation accuracy (i.e. with low $R^2(\theta_s)$), and splits them at points where the approximated signal changes abruptly. Then new functions are learned for two new states, using portions of continuous data that correspond to those states (Step 3b in Figure 6.3). This operation also accounts for the autonomous jumps in a system (i.e. it models significant changes in continuous signals). Abrupt changes in a signal are detected using the wavelet transform. Due to complexity and importance of this step, it is described separately in the following two sections.

6.4 Abrupt Change Detection

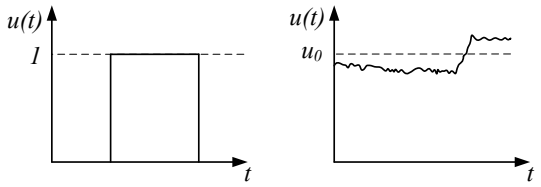
Our approach for learning behavior models of hybrid systems presented in this thesis is partially based on detecting abrupt changes in system continuous signals. Therefore, a brief review of several techniques for abrupt change detection is given in this section.

As explained earlier, hybrid systems comprise both discrete and continuous dynamics. These dynamics induce two types of changes in a system [NB02]:

Controlled jumps: Changes in discrete control signals imposed by a control system are called controlled jumps. These changes trigger state changes of the actuators, which further change the state of the continuous physical system.

Autonomous jumps: Continuous process variables of a system can abruptly change the state of a system without the influence of a control system. This happens due to threshold crossings or some external influences (e.g. disturbances or human factors). Such changes are called autonomous jumps.

An example of these changes is given in Figure 6.7. Contrary to control jumps, which are normally present in every hybrid production system, autonomous jumps are often artificially introduced to ease the modeling tasks, especially when complex nonlinear dynamics is present in the system [MB00]. Moreover, while control jumps are predefined and can be used in a form of a list of symbols that trigger transitions between system states, autonomous jumps need to be detected first. Their detection represents a specific challenge in many tasks, including learning behavior models.



(a) Controlled jump. (b) Autonomous jump.

Fig. 6.7 Two types of jumps in a hybrid production system.

In the previous two decades, a significant research has been done on detecting abrupt changes, which represent autonomous jumps in production systems. This research resulted in a number of various approaches. These are roughly divided in linear model-based approaches (parameter estimation for a given model, see e.g. [BN93]), model-free approaches (such as support vector machines for abrupt change detection [DD03]), and non-parametric approaches (*Discrete Fourier Transform* [OS09] and *Discrete Wavelet Transform* (DWT) [AAB09, Uv08]). A more detailed classification can be found in [Uv05]. These approaches are used in a wide range of applications, including condition monitoring, seismic data processing, quality control, image processing, and prediction of natural disasters [BN93]. DWT is often used for segmenting continuous signals, particularly for purposes of fault detection and diagnosis (please see [Uv05, GCMC00, ZY01]).

In our model-learning endeavor, we assume no a priori knowledge about the types of existing signals, thus the model-based approaches cannot be applied. Due to the effectiveness, speed, and popularity of DWT in analyzing both sinusoidal and impulse, high-frequency and low-frequency signal components, we have applied it to detecting autonomous jumps in continuous process variables. This step is an integral part of our comprehensive approach for learning behavior models of hybrid production systems. It is explained in the following section.

6.5 Modeling Autonomous Jumps with State Splits

In this section, we describe how the HyBUTLA algorithm was extended to model the autonomous jumps in the continuous part of a hybrid system. We give details of the abrupt change detection and provide the recursive *split* function. State split criteria are presented in detail and the proof for the benefits of the *split* function is given.

The following text is based on the work that we published in [Vod12] and [Vod13].

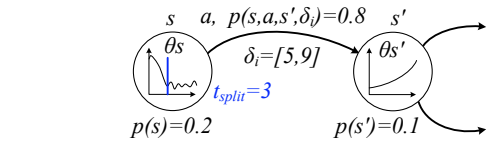
The initial version of the HyBUTLA algorithm [VKBNM11a] suffered from the following drawbacks:

1. **Neglected autonomous jumps:** Although continuous signals in the real systems often include abrupt changes, these are not taken into account by the algorithm. New states are derived solely based on controlled jumps.
2. **Insufficient runtime performance:** In order to adequately approximate nonlinear functions that can change abruptly, the advanced and time-consuming regression methods are often needed.

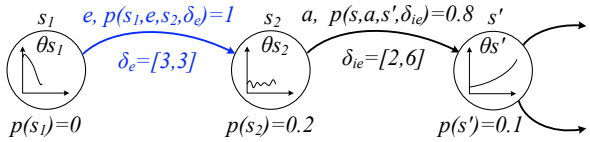
3. **Dependence of state merging on approximation functions:** The additional criteria for state compatibility that compared similarities between θ functions of states candidates for merging had to be defined differently based on the type of these functions. As stated in the previous section, if multiple linear regression is used, it is checked if the corresponding regression coefficients of functions from the two states are similar enough. For neural networks, the weights and biases of neurons would have to be compared, etc.

In [Vod12] we have extended the HyBUTLA algorithm to account for autonomous jumps originating in continuous signals. When an abrupt change is detected in some automaton state, this state is split into two states at the moment when that change occurs. Such moments can be detected using discrete wavelet transform. This enabled simpler and much faster regression methods, such as multiple linear regression, to reach high approximation accuracy, even when modeling nonlinear systems. Moreover, modeling autonomous jumps excluded the need to compare θ functions of two states before merging.

The effects of the state split are illustrated in Figure 6.8. Figure 6.8(a) shows the automaton before applying the *split* function. The continuous dynamics in the state s comprises two different trends, which are approximated by the function θ_s. Using discrete wavelet transform described later on, a point in time t_{split} has been detected when the abrupt change between these trends takes place. The state is then split at this particular point (given that other criteria explained later in this section are fulfilled), resulting in two new states s_1 and s_2. Figure 6.8(b) shows the result of the splitting. A new symbol e is generated that triggers the transition (with probability one) between two newly created states. A new transition timing constraint δ_e is created based on the timing of detected abrupt change, and new functions θ_{s_1} and θ_{s_2} are learned.



(a) Automaton before splitting.



(b) Automaton after splitting.

Fig. 6.8 The effects of the *split* function.

6.5.1 Discrete Wavelet Transform

As already mention in Section 6.4, non-parametric approaches to abrupt change detection include *Fourier Transform* (FT) and *Wavelet Transform* (WT). In contrast

to FT, which represents the signal as the sum of sinusoids of different frequencies and gives the amplitude-frequency relation, WT preserves the time information of the original signal. The Fourier Transform is given by:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt, \tag{6.6}$$

where $f(t)$ is the transformed signal. The *Continuous Wavelet Transform* (CWT) breaks the signal into the sum of scaled and shifted versions of the so-called *mother wavelet* $\psi(t)$. A mother wavelet is basically a signal of limited length and zero average value. The CWT of the signal $f(t)$ is calculated as follows [Uv05]:

$$CWT(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(t)\psi^* \left(\frac{t-b}{a}\right) dt, \tag{6.7}$$

where $\psi^*(t)$ is a complex conjugate of the mother wavelet, and $b, a \in \mathbb{R}, a \neq 0$ are the shifting and the scaling parameters, respectively. The parameter b represents the time parameter, while a relates to the frequency. By changing b values, a mother wavelet $\psi(t)$ is shifted over the signal $f(t)$. Changes in parameter a stretch or compress the mother wavelet, thus enable detection of both slow and fast components of the signal. Higher values of $CWT(a,b)$ coefficients indicate higher correlation with $f(t)$.

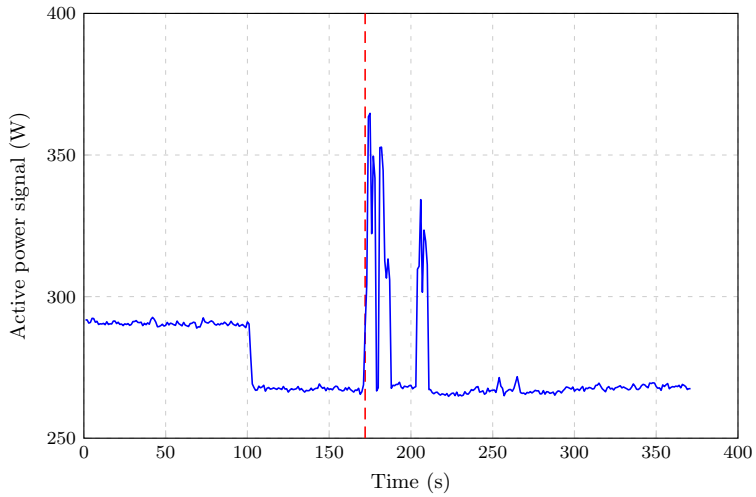
Various forms of the mother wavelet $\psi(t)$ exist. The simplest one that is often used in abrupt change detection is the *Daubechies 1* wavelet, also known as db1 or the *Haar* wavelet [Dau92]. This wavelet was used as an asset to the HyBUTLA algorithm. It is defined as:

$$\psi(t) = \begin{cases} 1 & \text{if } t \in [0, 0.5) \\ -1 & \text{if } t \in [0.5, 1) \\ 0 & \text{if } t \notin [0, 1). \end{cases} \tag{6.8}$$

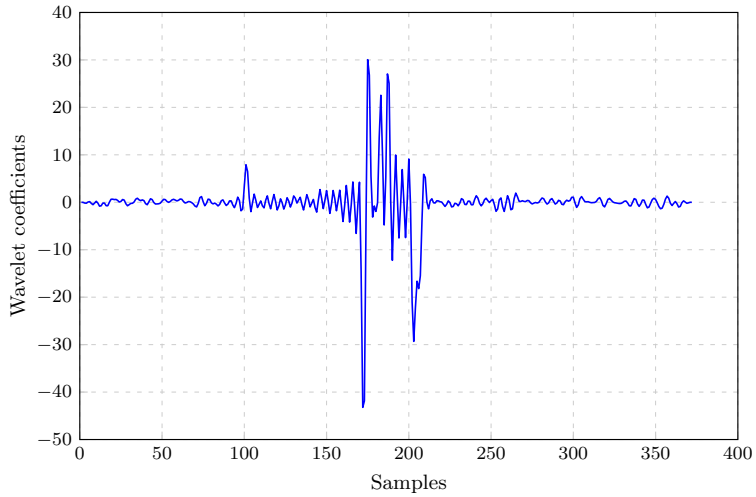
The calculation of CWT for all values of a and b over all time is very computationally inefficient. Therefore, the much faster *Discrete Wavelet Transform* (DWT) [Dau92, AAB09, Uv05] is used instead. It is calculated only for a limited number of a and b values. By inserting $a = a_0^m$, $b = ha_0^m b_0$ and $t = kT$ in expression (6.7), where $k, h, m \in \mathbb{N}^+$, the expression for calculating DWT is obtained:

$$DWT(m,h) = \frac{1}{\sqrt{a_0^m}} \sum f(kT)\psi^* \left(\frac{kT - ha_0^m b_0}{a_0^m}\right). \tag{6.9}$$

An example of the abrupt change detection in a signal using DWT is given in Figure 6.9. It shows one active power signal (Figure 6.9(a)) measured in a real small process plant (presented in Section 8.2) and its discrete wavelet transform (Figure 6.9(b)) calculated using the Haar wavelet. It can be seen that the largest coefficient in the absolute value (obtained at the sample 172 in Figure 6.9(b)) corresponds to the



(a) An active power signal recorded in a real plant and its biggest abrupt change (dashed red line).



(b) DWT of the given signal.

Fig. 6.9 A real-world signal and its Discrete Wavelet Transform.

biggest abrupt change in the signal (marked by a dashed red line in Figure 6.9(a)). This fact is exploited by the *split* function.

6.5.2 The split Function

When a new state $s \in S$ of the automaton is generated either based on controlled or autonomous jump, its θ_s function (see Definition 13) for approximating the behavior of the continuous output signal(s) needs to be learned. To this aim, various

regression methods can be used that take the values of the continuous input signals $u \in U$, continuous output signals $y \in Y$, and time t as inputs. The measure of deviation between the learned function θ_s and the real values y_j of the approximated output signal y is expressed as the coefficient of determination, already given by the expression (6.1). The criteria that have to be fulfilled in order to split the state s are as follows:

1. **Low $R^2(\theta_s)$ value:** The lowest acceptable $R^2(\theta_s)$ value is given by threshold σ. In case $R^2(\theta_s) < \sigma$, the state s is a candidate for splitting.
2. **Avoiding *dummy* states:** Dummy states are states where θ_s function is trivial, i.e. that contain only two y_j points for learning the function (simple linear regression would result in $R^2(\theta_s) = 1$). Allowing the creation of dummy states could result in a rapid increase of their number, which is contradictory to the general requirement of having the model as small as possible. Therefore, the split is performed only if it would generate two new states with each having at least three data points for learning the θ_s function.
3. **Positive transition timings after split:** When discrete wavelet transform identifies the time t_{split} as a point of splitting, this time has to be subtracted from δ intervals of all transitions that leave the state s. If all these intervals remain positive, the split takes place. This situation is already shown in Figure 6.8. If in this particular case would be $t_{split} = 6$, we would have $\delta_{ie} = [-1, 3]$ (i.e. a negative timing) which is not allowed by Definition 13.
4. **Increased model average R^2:** Splitting the states could decrease the average R^2 of the model. Therefore, it is preformed only if it brings the global increase in the function approximation accuracy (i.e. if the average R^2 gets increased). In [Vod12], we gave this requirement as the theorem with the proof for its necessary condition. Subsection 6.5.3 gives proofs for both necessary and sufficient conditions, which we originally published in [Vod13].

The recursive *split* function is given in Figure 6.10. It receives the already created automaton A that models only controlled jumps. In addition, a threshold value σ is given for the lowest allowed R^2 value of the function θ_s in the state s. The function $split$ outputs a new automaton A_{split} that models both controlled and autonomous jumps. At first, an automaton A_{split} is initialized with the elements of the automaton A (line 0). Then the state s whose θ_s function does not satisfy the threshold σ is found (line 1, see criterion 1 for the state split above). Discrete wavelet transform of the output signal $y \in Y$ used for learning that θ_s function is calculated (line 2) and its coefficients are iterated (line 3). The largest (in the absolute value) coefficient c_k is found (line 4). Then the state s is split into the states s_1 and s_2 at the point t_{split}, which denotes the relative timing of occurrence of coefficient c_k (line 5). Ending state probabilities are associated with new states, as well as the new symbol e that triggers a transition τ_e between them (lines 6-7). The probability of this transition is $p(s_1, e, s_2, \delta_e) = 1$ and its timing constraint $\delta_e = [t_{split}, t_{split}]$ (lines 8 and 9). For newly created states, functions θ_{s_1} and θ_{s_2} are learned using their corresponding portions of continuous data (line 10). The *if*-condition in line 11 tests compliance with criteria 2 and 3 for the state split (see above). When s_1 and s_2 are not dummy states, and when the split results in positive transition timings of the transitions that leave the state s, the elements of the automaton A_{split} are updated (line 12). In case the criteria are not fulfilled for coefficient c_k and its timing t_{split} (line 13), the next largest coefficient of discrete wavelet transform is evaluated. The procedure is repeated for every state with insufficient R^2 value. Finally, if the automaton A_{split} is

Given:
(1) 1-SDHA $A = (S, s_0, \Sigma, T, \Delta, P, x, \Theta)$
(2) Predefined threshold σ for the lowest allowed R^2 value
Result: New automaton A_{split} that models abrupt changes
(0) $A_{split} = A$ % initialisation of the automaton A_{split}
(1) **for all** $s \in S \mid R^2(\theta_s) < \sigma$
(2) $C = DWT(y) \mid y = \theta_s(t, u)$ % calculate DWT, C is a set of DWT coefficients
(3) **while** $notEmpty(C)$
(4) $c_k = max(abs(C))$ % find the largest DWT coefficient in the absolute value
t_{split} is the relative time of the coefficient c_k.
(5) Create states s_1 and s_2 by splitting the state s at point t_{split}.
(6) Ending state probabilities of created states are $p(s_1) = 0$ and $p(s_2) = p(s)$.
(7) Generate the symbol e that triggers transition τ_e between s_1 and s_2.
(8) The probability of transition τ_e is $p(s_1, e, s_2, \delta_e) = 1$.
(9) The timing constraint of generated symbol e is $\delta_e = [t_{split}, t_{split}]$.
(10) Learn functions θ_{s_1} and θ_{s_2} for states s_1 and s_2, respectively.
(11) **if** (s_1 and s_2 are not dummy states) **and**
(intervals $\delta_{ie} = [t_{i1} - t_{split}, t_{i2} - t_{split}]$ are positive,
where $\delta_i = [t_{i1}, t_{i2}]$ are timings of all transitions $\tau_i \in T$ leaving the state s)
then
(12) Update automaton $A_{split} = (S_{split}, s_0, \Sigma_{split}, T_{split}, \Delta_{split}, P_{split}, x, \Theta_{split})$:
$S_{split} = (S \backslash \{s\}) \cup \{s_1, s_2\}$
$\Sigma_{split} = \Sigma \cup \{e\}$
$T_{split} = T \cup \{\tau_e\}$
$\Delta_{split} = (\Delta \backslash \{\delta_i\}) \cup \{\delta_e, \delta_{ie}\}$
$P_{split} = (P \backslash \{p(s)\}) \cup \{p(s_1, e, s_2, \delta_e), p(s_1), p(s_2)\}$
$\Theta_{split} = (\Theta \backslash \{\theta_s\}) \cup \{\theta_{s_1}, \theta_{s_2}\}$
(13) **else** $C = C \backslash \{c_k\}$
(14) **end if**
(15) **end while**
(16) **end for**
(17) **if** average $R^2_A <$ average $R^2_{A_{split}}$ **then**
(18) A_{split} = *split*(A_{split})
(19) **else** return A
(20) **end if**

Fig. 6.10 The recursive *split* function.

obtained whose average accuracy $R^2_{A_{split}}$ (over all states) is larger than the average accuracy R^2_A of the automaton A, the function split is recursively called providing A_{split} as an input (lines 17-18). Otherwise, if no better automaton than A could be created, or if no split has been performed, the automaton A is returned (line 19).

6.5.3 The Benefits of the split Function

As we argued in [Vod12], the *split* function needs to ensure both the local and global increase of the function approximation accuracy, expressed as the coefficient of determination R^2. When state $s_k \in S$ is split into states s'_k and s''_k, the local increase is ensured under the condition:

$$R^2(\theta_{s'_k}) > R^2(\theta_{s_k}) \text{ and } R^2(\theta_{s''_k}) > R^2(\theta_{s_k}). \tag{6.10}$$

The global increase is ensured when the following holds:

$$R^2(\theta_{s'_k}) + R^2(\theta_{s''_k}) > R^2(\theta_{s_k}) + R^2_A, \tag{6.11}$$

where R^2_A is the average coefficient of determination over w states of the automaton A, calculated as:

$$R^2_A = \frac{\sum_{i=0}^{w-1} R^2(\theta_{s_i})}{w}. \tag{6.12}$$

The conditions (6.10) and (6.11) can be better understood using the following example.

Example 4. *Let an automaton A be given that has four states: $S = \{s_0, s_1, s_2, s_3\}$ with the coefficients of determination respectively: $R^2(\theta_{s_0}) = 0.4$, $R^2(\theta_{s_1}) = 0.1$, $R^2(\theta_{s_2}) = 0.2$, and $R^2(\theta_{s_3}) = 0.3$. Their average value is $R^2_A = 0.25$. Assume the state s_2 is now split into states s'_2 and s''_2. A new automaton A_{split} is obtained that has five states, i.e. $S_{split} = \{s_0, s_1, s'_2, s''_2, s_3\}$. Let the corresponding coefficients of determination of the new states be $R^2(\theta_{s'_2}) = 0.21$ and $R^2(\theta_{s'_2}) = 0.22$. Since both values are higher than $R^2(\theta_{s_2})$, the local accuracy increase given by expression (6.10) is ensured, but not the global one since:*

$$R^2(\theta_{s'_2}) + R^2(\theta_{s''_2}) = 0.43 \not> R^2(\theta_{s_2}) + R^2_A = 0.45.$$

In this case, the split would be rejected. However, if the split of s_2 would result in states s'_2 and s''_2 such that $R^2(\theta_{s'_2}) = 0.21$ and $R^2(\theta_{s'_2}) = 0.26$, both conditions would be satisfied and the split would be performed (assuming other criteria given in the previous subsection are also satisfied).

We gave the following theorem in [Vod13], which proves that the expression (6.11) is the necessary and sufficient condition for ensuring the global increase of the function approximation accuracy.

Theorem 13. *Let $A = (S, s_0, \Sigma, T, \Delta, P, x, \Theta)$ be the hybrid automaton with $|S| = w$ states. Its average R^2 value is given by the expression (6.12). The split of a state $s_k \in S$ into the states s'_k and s''_k brings the global increase of the function approximation accuracy iff for the two resulting states s'_k and s''_k the condition given by the inequality (6.11) holds.*

Proof. After the state $s_k \in S$ is split, a new automaton A_{split} is obtained with the number of states $|S_{split}| = w + 1$. The global increase of the function approximation accuracy is obtained under the condition:

$$R^2_{A_{split}} > R^2_A. \tag{6.13}$$

The proof of the *if-part* of the theorem is given first (initially published in [Vod12]), i.e. prove (6.13) assuming (6.11). Expression (6.11) can be rewritten as:

$$R^2(\theta_{s'_k}) + R^2(\theta_{s''_k}) - R^2(\theta_{s_k}) - R^2_A > 0.$$

The R^2 values of the states unaffected by split are added and subtracted:

$$\sum_{i=0}^{k-1} R^2(\theta_{s_i}) + R^2(\theta_{s'_k}) + R^2(\theta_{s''_k}) + \sum_{i=k+1}^{w-1} R^2(\theta_{s_i})$$

$$- \sum_{i=0}^{k-1} R^2(\theta_{s_i}) - R^2(\theta_{s_k}) - \sum_{i=k+1}^{w-1} R^2(\theta_{s_i}) - R^2_A > 0.$$

The first line is the averaged R^2 for the automaton A_{split} multiplied by $(w+1) > 0$, thus it follows:

$$(w+1)R^2_{A_{split}} - \left[\sum_{i=0}^{w-1} R^2(\theta_{s_i}) + R^2_A\right] = (w+1)R^2_{A_{split}} - (w+1)R^2_A > 0.$$

When this inequality is divided by $(w+1) > 0$, it is finally obtained:

$$R^2_{A_{split}} - R^2_A > 0. \tag{6.14}$$

The expressions (6.13) and (6.14) are equal, thus the *if-part* of the theorem is proven. We now prove the *only-if-part* of the theorem: prove (6.11) assuming (6.13). Two average values from the inequality (6.13) can be subtracted as follows:

$$\frac{\sum_{i=0}^{k-1} R^2(\theta_{s_i}) + R^2(\theta_{s'_k}) + R^2(\theta_{s''_k}) + \sum_{i=k+1}^{w-1} R^2(\theta_{s_i})}{w+1} - \frac{\sum_{i=0}^{w-1} R^2(\theta_{s_i})}{w} =$$

$$\frac{w\left[\sum_{i=0}^{k-1} R^2(\theta_{s_i}) + R^2(\theta_{s'_k}) + R^2(\theta_{s''_k}) + \sum_{i=k+1}^{w-1} R^2(\theta_{s_i})\right] - (w+1)\sum_{i=0}^{w-1} R^2(\theta_{s_i})}{w(w+1)} =$$

$$\frac{w\left[\sum_{i=0}^{k-1} R^2(\theta_{s_i}) + R^2(\theta_{s'_k}) + R^2(\theta_{s''_k}) + \sum_{i=k+1}^{w-1} R^2(\theta_{s_i})\right]}{w(w+1)} -$$

$$\frac{w\left[\sum_{i=0}^{k-1} R^2(\theta_{s_i}) + R^2(\theta_{s_k}) + \sum_{i=k+1}^{w-1} R^2(\theta_{s_i})\right]}{w(w+1)} - \frac{\sum_{i=0}^{w-1} R^2(\theta_{s_i})}{w(w+1)} > 0.$$

The last expression can be rewritten in the following way:

$$\frac{w\left[R^2(\theta_{s'_k}) + R^2(\theta_{s''_k}) - R^2(\theta_{s_k})\right]}{w(w+1)} - \frac{R_A^2}{w+1} > 0,$$

which is (multiplied by $(w+1) > 0$) equal to the expression (6.11). □

6.6 Algorithm Properties

In Section 4.4, it has been shown that 1-SDHAs cannot be learned in the sense of strong polynomial identification in the limit with probability one (StrongPolyIDLimitProb1). However, here we prove that our HyBUTLA algorithm satisfies the criteria of weak polynomial identification in the limit with probability one (WeakPolyIDLimitProb1), i.e. we show that it identifies 1-SDHAs in the limit with probability one in time polynomial in the size of the input data (see Definition 19 in Section 4.3).

In order to achieve WeakPolyIDLimitProb1 of 1-SDHAs, our algorithm basically needs to ensure the following:

1. **Convergence:** (i) identification of automaton states S and transitions T (i.e. its structure), (ii) identification of symbol probabilities and ending state probabilities P and (iii) identification of transition timings Δ.
2. **Polynomial runtime:** algorithm runtime must be polynomial in the size[3] n of the input sample $\mathcal{D}$. Since the HyBUTLA algorithm learns Θ functions that approximate the continuous dynamics of a system, we impose the additional requirement on the algorithm to adequately approximate the continuous output signals by these functions, and moreover, in polynomial time.

Lemma 3. *The HyBUTLA algorithm identifies the correct automaton structure (states and transitions) in the limit with probability one.*

Proof. Following [CO94], [CO99] and [HT00], the HyBUTLA algorithm uses the same statistical measure for deciding if two states are compatible, i.e. this measure is based on the Hoeffding bound (expression (6.2)) and implemented as the function *fractions-different* (expression (6.3)). Therefore, in this proof we summarize and extend the results given in these papers.

The probability that any of the two estimations $\frac{f_0}{g_0}$ and $\frac{f_1}{g_1}$ in the function *fractions-different* is wrong is bounded by α. Thus, the probability that the function works with wrong estimations is at most 2α.

By t_n we denote the number of the PTA states obtained after receiving the input sample $\mathcal{D}$ of size n. The function *compatible* (see Figure 6.6) compares the transition probabilities, but also the ending state probabilities for a pair states. Since there are at most t_n states and $t_n - 1$ transitions in a subtree, the function *fractions-different* is called not more than $2t_n$ times in one iteration of the HyBUTLA algorithm (i.e. in one call of the function *compatible*).

The total probability that any performed test in the function *compatible* will fail is thus limited by $2\alpha \cdot 2t_n$.

[3] The sample size can be expressed either as a number of examples, or as a total sum of lengths of the examples [dlH06]. The given results hold for both cases.

As argued in [HT00], the given probability bound is valid only when the result of the test implemented in the function *fractions-different* is independent from other tests. This is obviously not the case, as some previous successful test could have already used at least one state from the considered pair (candidates for merging). The probability that this previous test was incorrectly successful is also bounded by 2α. In this case, the total probability that any performed test that uses the incorrect result of a previous test will fail is thus limited by

$$p(E_n) = (2\alpha + 2\alpha)2t_n, \tag{6.15}$$

where E_n is an error event.

Since the number of tests grows with n, we allow the parameter α to depend on n and denote it by α_n. The parameter α_n will be chosen so that the sum $\sum_{n=1}^{\infty} n\alpha_n$ is finite (as in [CO99]). The number of PTA states t_n can grow at most linearly with n, thus $t_n = cn$, where $c \in (0, 1]$ is a constant. The same holds for the number of examples that arrive at any of considered states, i.e. $g_0 = c_0 n$ and $g_1 = c_1 n$, $c_0, c_1 \in (0, 1]$ are constants. It follows:

$$p(E_n) = 8\alpha_n t_n = 8cn\alpha_n. \tag{6.16}$$

By the Borel-Cantelli lemma [Fel50], if $\sum_{n=1}^{\infty} p(E_n) < \infty$, then with probability one only finitely many events E_n exist[4]. By choosing $\alpha_n = \frac{k}{n^2}$, where $k > 0$ is a constant (similar to [HT00]), we can write:

$$\lim_{n\to\infty} p\left(E_n\right) = \lim_{n\to\infty} \left(8cn\frac{k}{n^2}\right) = 0. \tag{6.17}$$

The probability that the function *compatible* returns a wrong result goes to zero as n grows. Regardless of how many previous (dependent) tests might have been wrong, the probability of the error event always goes to zero in the limit. It is important to note that the function *compatible* is in the HyBUTLA algorithm called only a finite number of times.

To complete the proof it is also important to consider the test of similarity between time intervals $\delta \in \Delta$ of transitions that passed the compatibility check given by the function *fractions-different*. This simple test is deterministic, i.e. it comes down to checking whether intervals belong to the same time cluster (line 12 in Figure 6.6). When this is the case, the states will be merged. The similarity test for time intervals does not depend on any probabilistic parameters (such as α_n), i.e. it never inserts an error in the merging process. Thus, the lemma is proven. □

Observation 7. *Carrasco and Oncina [CO94] classify the errors of the compatibility test based on the Hoeffding bound in the following way:*

- *Type I error: Two compatible states are not merged.*
- *Type II error: Two non-compatible states are merged.*

It is easy to see how both of these errors disappear when the number of examples n grows.

The global probability of type I error $p_g(\mathit{TypeI})$ for the whole automaton, assuming independent tests, is bounded by:

[4] No assumption of the independence of those events is required by Borel-Cantelli lemma.

$$p_g(\mathit{TypeI}) = 2\alpha_n(|\Sigma|+1)t_n. \tag{6.18}$$

During the merging process, the algorithm makes at most $\frac{1}{2}t_n(t_n-1)$ *comparisons of the states. By* m *we denote the number of states of the final automaton. If type II error happened, in the worst case there were in total* $\frac{1}{2}m(m-1)(|\Sigma|+1)$ *wrong comparisons. Following the expression (6.3), the error of each comparison is given as:*

$$\mathit{error} = \sqrt{\frac{1}{2}\log\left(\frac{2}{\alpha_n}\right)}\left(\frac{1}{\sqrt{g_o}}+\frac{1}{\sqrt{g_1}}\right), \tag{6.19}$$

thus the global probability of type II error $p_g(\mathit{TypeII})$ *is bounded by:*

$$p_g(\mathit{TypeII}) = \frac{1}{2}m(m-1)(|\Sigma|+1)\cdot \mathit{error}. \tag{6.20}$$

Now if we pick α_n *as in the proof above so that* $\alpha_n = \frac{k}{n^2}$*, knowing that* $t_n = cn$*,* $g_o = c_0 n$ *and* $g_1 = c_1 n$ *(*$k > 0, c, c_0,\ c_1 \in (0,1]$*), it is easy to see that both* $p_g(\mathit{TypeI})$ *and* $p_g(\mathit{TypeII})$ *converge to zero as* n *grows.*

Observation 8. *The choice of the* α_n *parameter has significant influence on the convergence speed. By choosing* $\alpha_n = \frac{k}{n^2}$*,* $g_o = c_0 n$ *and* $g_1 = c_1 n$ *(*$k > 0, c_0, c_1 \in (0,1]$*), the error expression (6.19) becomes:*

$$\mathit{error}_1 = \sqrt{\frac{1}{2}\log\left(\frac{2n^2}{k}\right)}\left(\frac{1}{\sqrt{c_0 n}}+\frac{1}{\sqrt{c_1 n}}\right). \tag{6.21}$$

For taking $\alpha_n = \frac{k}{n^3}$*, we similarly get:*

$$\mathit{error}_2 = \sqrt{\frac{1}{2}\log\left(\frac{2n^3}{k}\right)}\left(\frac{1}{\sqrt{c_0 n}}+\frac{1}{\sqrt{c_1 n}}\right). \tag{6.22}$$

When n *grows, the first error expression converges to zero faster than the second one. Figure 6.11 illustrates these cases for* $k = c_0 = c_1 = 1$ *and* $n = 1000$*. This means that, as we increase the value of the* α_n *parameter, the convergence speed increases as well.*

Lemma 4. *The HyBUTLA algorithm identifies the correct transition probabilities of a 1-SDHA in the limit with probability one.*

Proof. In this proof, we use a common measure of distance between two probability distributions, known as *Kullback-Leibler* (K-L) divergence or *relative entropy* (it has also been used as a measure of divergence between two automata in the MDI algorithm, see Subsection 2.4.3). For two probability distributions P_1 and P_2, K-L divergence is defined as:

$$d_{KL}(P_1||P_2) = \sum_i P_1(i)\log\frac{P_1(i)}{P_2(i)}. \tag{6.23}$$

When rolling a die, a probability of getting any of the 6 outcomes j is $p_j = 1/6$. Thus if the number of runs is denoted by m it holds for all j:

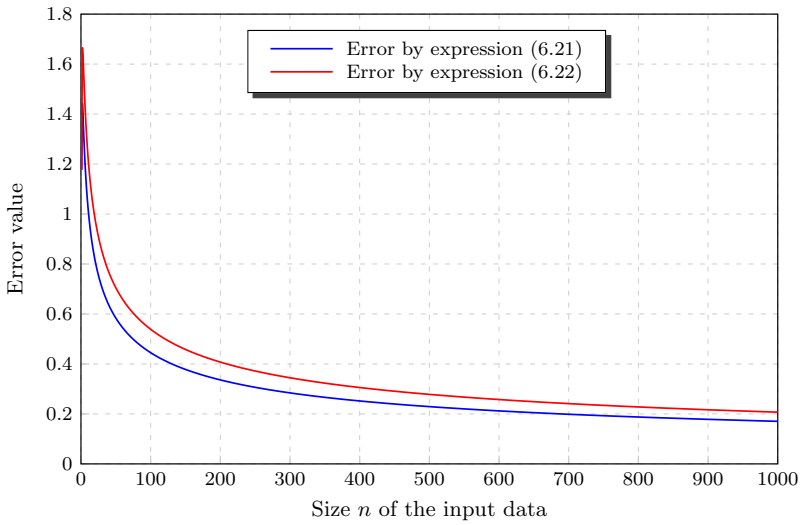


Fig. 6.11 Convergence speed for different selection of α_n. Convergence is faster for larger values of α_n.

$$\lim_{m\to\infty} p_j(m) = p_j. \tag{6.24}$$

In the sense of K-L divergence, when the number of runs m grows, the estimated probability P_2 comes closer to the actual probability P_1 and eventually $d_{KL}(P_1||P_2)$ goes to zero. However, this does not hold when the number of possible outcomes j is infinite ($p_\infty = 1/\infty = 0$).

In the context of identifying transition probabilities of a 1-SDHA, the expression (6.23) gets the following form:

$$d_{KL}(A||A') = \sum_{z\in\Sigma^*} p(z|L_A) \log \frac{p(z|L_A)}{p(z|L_{A'})}. \tag{6.25}$$

Similarly to [CO99], we define the so-called *support* of the language L_A as a subset $R(L_A) = \{z \in \Sigma^* : p(z|L_A) > 0\}$. Since $R(L_A)$ is infinite in general case, it includes those examples z whose probability $p(z|L_A) > 0$ is arbitrarily small. When a finite input sample is given, it might not include such examples so their probabilities will be incorrectly estimated as zero. Therefore, the expression (6.25) would obtain large value.

However, if the structure of the automaton A is identified in the limit with probability one, i.e. from a sequence of positive examples $\mathcal{D}$ (see Lemma 3), all such small probabilities can be correctly estimated by counting the numbers of examples that visit, leave and end in corresponding states. In this case, the K-L divergence $d_{KL}(A||A')$ will eventually converge to zero. This means that the transition probabilities, as well as the ending state probabilities, will be estimated correctly in the limit with probability one. □

The means of estimation of the transition and ending state probabilities are given by the following corollary.

Corollary 8. *Based on the previous lemma, a transition probability as well as an ending state probability are correctly estimated using expressions (6.4) and (6.5), respectively.*

Lemma 5. *The HyBUTLA algorithm identifies the correct transition timings.*

Proof. Assume there are two transitions $\tau' = (s, a, s', \delta')$ and $\tau'' = (s, a, s'', \delta'')$ that should be merged after merging their corresponding states (states were merged according to the compatibility criteria of the function *compatible* given in Figure 6.6). Let their time intervals be $\delta' = [t'_1, t'_2]$ and $\delta'' = [t''_1, t''_2]$. The time interval of the resulting transition is calculated according to the following:

$$\delta = [min(t'_1, t''_1), max(t'_2, t''_2)]. \tag{6.26}$$

In this way, the resulting δ interval always includes the time spans of both transitions, thus no wrong timing is included. □

The following two lemmas consider the runtime properties of the HyBUTLA algorithm.

Lemma 6. *The HyBUTLA algorithm learns functions $\theta_s \in \Theta, \forall s \in S$ of a 1-SDHA by the means of an approximation method $\mathcal{M}$ within predefined marginal error ε in polynomial time.*

Proof. In the proof of Lemma 1 in Section 3.5, it has been shown that a method $\mathcal{M}$ exists with the following properties:

$$error(\mathcal{M}(\theta_s)) \leq \varepsilon \text{ and}$$

$$runtime(\mathcal{M}(\theta_s)) \in O(Poly)$$

for some polynomial $Poly$ and $\theta_s \in \Theta, \forall s \in S$ are the functions of a 1-DHA. Since these functions have no specific properties comparing to those of 1-SDHAs (see Definition 4 and Definition 13), the proof of Lemma 1 proves this lemma as well. □

As stated in Section 3.5, the value of the predefined marginal error ε depends on the application area. The allowed ε values for approximated continuous output signals are obtained by analyzing the modeled system and using the expert knowledge.

Observation 9. *Depending on the type of continuous output signals $y \in Y$ that need to be approximated, sometimes the faster methods also capable of reaching marginal error ε in polynomial time could be used. For example, for piecewise linear signals, linear regression could be used etc.*

Lemma 7. *The HyBUTLA algorithm runs in time polynomial in the size of the input data n.*

Proof. (Rework of the proof that we published in [NSV+12]) After receiving input sample $\mathcal{D}$ of size n, a prefix tree is constructed with t_n states. Then the Θ functions are learned for every state. These actions are done in Step 1 in Figure 6.3, and they run in $O(t_n)$. In Step 2, the HyBUTLA algorithm compares t_n^2 states in the worst case. Function *compatible* (see Figure 6.6) recursively checks compatibility of corresponding child states. Then the compatible states are merged, the automaton is

determinized and the Θ functions are learned for newly created states. Thus the total runtime of this step is $O(t_n^3)$.

Now we analyze the time complexity of the additional Step 3 for splitting the states. Splitting is based on discrete wavelet transform, whose time complexity is $O(N)$, where N is the number of data points for its calculation [GB97]. The maximum number of states to be split in one iteration of the *split* function is the size of the prefix tree t_n (worst case when no merges have happened). There is a finite number of its iterations. DWT is calculated for each of the t_n states, thus it runs in $O(t_n)$. The Θ functions are once again learned for the states created in this step of the algorithm. According to Lemma 6, these functions are learned in polynomial time.

The number of PTA states is linear in the size of the input sample, i.e. $t_n = cn$, where $c \in (0, 1]$ is a constant. Therefore, the overall time complexity of the HyBUTLA algorithm is $O(n^3)$. □

Theorem 14. *The class of 1-SDHAs is WeakPolyIDLimitProb1.*

Proof. The result follows from Lemmas 3, 4, 5, 6, and 7. □

Corollary 9. *1-SDHAs are IDLimitProb1. This result follows from Lemmas 3, 4, 5, and 6.*

Observation 10. *Please note that the class of 1-SDHAs is polynomially reachable. This means that a polynomial $Poly$ exists such that for any state s from any 1-SDHA A, there exists a learning example D_i with $|D_i| \leq Poly(|A|)$, such that D_i reaches s in A [VdWW08]. The papers [AD94] and [HKPV98] give the following decidability preconditions for the reachability problem of hybrid automata: (i) decoupled output variables, (ii) initialization after flow changes (i.e. after state transitions) and (iii) having only one clock. As we emphasized in [NSV+12], the class of 1-SDHAs fulfills these conditions as follows: (i) the output variables are decoupled based on the available expert knowledge, (ii) Θ functions implicitly reinitialize these variables and (iii) only one clock is used that is also reinitialized after every transition (i.e. the clock is reset).*

6.7 Conclusion

In this chapter, we presented our approach for automated learning of behavior models for hybrid production systems from data. We have described all stages of this approach, including data acquisition, noise removal and the learning algorithm. Major contributions of the chapter are:

1. We gave the HyBUTLA algorithm that is, to the best of our knowledge, the first algorithm for learning hybrid automata models, which can adequately represent characteristics of a hybrid production system (i.e. 1-SDHA models). The models are learned automatically from data recorded in a system.
2. We proved the convergence and polynomial runtime properties of the HyBUTLA algorithm. This algorithm learns 1-SDHA models in the limit with probability one from positive examples (text) only. Moreover, it is a polynomial-time algorithm, i.e. it learns 1-SDHAs in time polynomial in the size of the input data.

3. Our research has shown that the successful application of the HyBUTLA algorithm depends on the following expert knowledge:

   - the structure of a system needs to be known (i.e. the parallelism model given by Definition 3 in Subsection 2.1.1 must exist) and represented as a set of sequential components that can be individually modeled by the HyBUTLA algorithm,
   - for each modeled component, sets of its discrete signals Q, continuous input signals U and continuous output signals Y need to be known (see Definition 13 in Section 4.2),
   - every used learning example needs to comprise logs (measurements) of all aforementioned signals.

Please note that we want to learn a model of normal system behavior. Therefore, we use only logs of normal production cycles, i.e. the positive learning examples. We assume that all these examples are correctly labeled as normal by human experts. This assumption is realistic in real-world hybrid production systems. Based on product specifications and its characteristics, the experts can easily say if the corresponding production cycle was normal or not. When, for any reason, some learning examples are mistakenly labeled as positive, our approach will not produce a reliable model of normal behavior.

The HyBUTLA algorithm however does not overcome the obstacle given by Corollary 5 in Section 4.4 that 1-SDHAs cannot be learned from data, whose size is polynomially linked with the size of the target automaton (i.e. HyBUTLA does not learn under the strong identification criteria given by Definition 20 in Section 4.3). Finding a special type of 1-SDHAs that can be potentially learned in the strong sense still remains a challenge for future work.

Chapter 7

Anomaly Detection Based on Learned Behavior Models

In the previous chapter, we presented our HyBUTLA algorithm that identifies behavior models for hybrid production systems automatically from data. In this chapter, we give an approach for using these models in the anomaly detection application. This approach serves to demonstrate that the models learned by the HyBUTLA algorithm are really usable in solving complex safety-related tasks of real-world production systems. The main results are summarized as follows:

- we give the ANODA algorithm that uses automatically learned behavior models for model-based anomaly detection,
- we provide a condition under which the anomaly detection system that uses the ANODA algorithm is the real-time system and can be used for online system monitoring,
- the expert knowledge needed for the ANODA algorithm to be successful is identified.

While here we give only the algorithm and show its real-time property, the conducted experiments are given in the following chapter.

We have organized this chapter in several sections. First, we explain the basic principle of model-based anomaly detection that uses behavior models learned with the HyBUTLA algorithm in Section 7.1. Then, in Section 7.2 we give our two definitions that ease the introduction to common types of anomalies in hybrid production systems. We have also cited Hawkins' definition of an outlier, which focuses our attention on detecting the abnormal behavior of a system. In addition, we describe control sequence, timing and process variable anomalies in a system. Finally, Section 7.3 presents our ANODA algorithm, as well as six specific fault types that the algorithm detects. As the algorithm works in an online manner (in parallel with the running system), it is important that it satisfies the properties of a real-time system. In Section 7.4 we recall definitions of real-time systems and prove the condition under which the anomaly detection system, that uses our ANODA algorithm, is a real-time system (Theorem 15). Section 7.5 concludes the chapter and lists the expert knowledge that the ANODA algorithm requires as an input. Parts of this chapter are based on our work published in [VKBNM11b] and [FFP+12].

7.1 The Principle of Model-Based Anomaly Detection

In this section, we illustrate the principle of model-based anomaly detection that relies on learned behavior models.

One of the most important tasks in modern industrial systems is to ensure the high level of their reliability and safety, which is also of the greatest importance for their productivity and efficiency. This task is becoming increasingly challenging, due to the rapid evolution and increasing complexity of such systems. However, various approaches have been developed over years to address this challenge (see Section 2.5), many of which are based on behavior models.

Automatic identification of behavior models can be positioned in the larger picture that also includes the application of those models in the model-based anomaly detection. As illustrated in Figure 7.1, the behavior model of a system can be learned using the HyBUTLA algorithm in an offline manner, assuming enough learning examples are provided. Once the reliable model is obtained, it can be used in parallel with the system during its runtime, i.e. in an online manner. The set of inputs that the system receives is also provided to its behavior model. The model calculates the prediction of the output, which can be both the value of the continuous output signal and the value of the discrete signal and its timing. The anomaly detection algorithm compares such a prediction with the observation coming from a system in the real-time. In case any significant discrepancy is detected, the anomaly (fault) is signaled to the operators of the system, which conduct a necessary action.

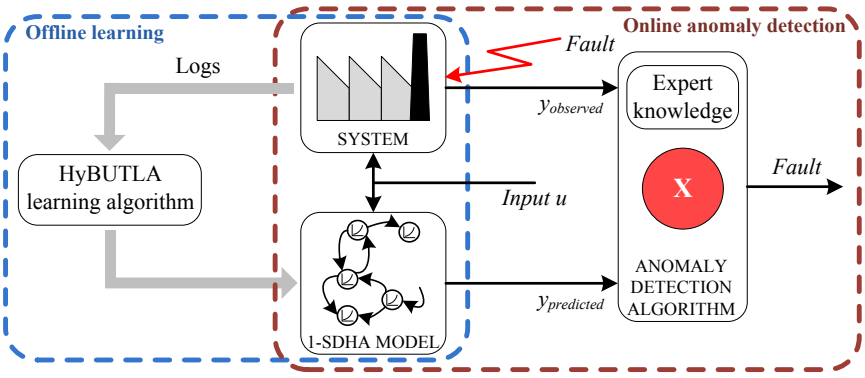


Fig. 7.1 Anomaly detection based on models learned with the HyBUTLA algorithm.

7.2 Anomalies in Hybrid Production Systems

Here we recall basic definitions of interest for anomaly detection, and explain three typical anomaly types that occur in hybrid production systems.

Generally speaking, the purpose of the anomaly detection is to find the anomalous (i.e. unlike, unexpected) objects, usually referred to as the outliers, which are caused by the faulty operating conditions. To be able to present types of these objects, we give the following two definitions, that we initially published in [VKBNM11b].

Definition 24 (Path Through the Automaton). Let $A = (S, s_0, \Sigma, T, \Delta, P, x, \Theta)$ be a hybrid automaton, according to Definition 14. A path PA through the automaton is defined as a sequence of transitions, i.e. $PA \subseteq T$.

Definition 25 (Observation of a System). An observation of a system is defined as a tuple $o = (a, t, \mathbf{u}, y)$, where:

- $a \in \Sigma$ is the symbol that triggers a transition,
- t is a relative time value (relative to the last control signal change),
- $\mathbf{u}$ is the vector of values of the continuous input signals, which is used for predicting the output value y_p,
- y is the observed value of the continuous output signal, which is compared with the predicted output y_p.

Anomalies in complex systems can have various causes. It is well-known that every dataset coming from a real-world running entity (such as a component of a process plant) comprises a so-called natural (normal) variability. For example, while most of the measurements of some constant signal under the same normal operating conditions have similar values, sometimes a measurement appears whose value is "far away" from the others. Based on such a measurement, the anomaly detection algorithms could wrongly recognize a faulty condition of a system. This data variability can be created by the imperfections in sensors, measurement noise, external disturbances or a human factor. Anomaly detection algorithms often need to be adjusted to account for this phenomenon. We are however interested in detecting the real anomalies, i.e. those that are caused by the faulty operating conditions appearing in a system during its runtime. These are defined by a statistician D. Hawkins.

Definition 26 (Hawkins' Definition of an Outlier [Haw80]). An outlier is an observation that differs so much from other observations as to arouse suspicion that it was generated by a different mechanism.

Anomalies in hybrid production systems can originate in both discrete control system, as well as in the continuous physical system. Three common types of anomalies are explained in the following.

Control sequence anomaly: In production facilities, the process is controlled by a control system, which emits discrete control signals such as the signal to open or close a valve. After emitting such control signal, the control system receives the feedback signal within some predefined time interval. This feedback carries the information was the action successfully executed or not. The *control sequence anomaly* occurs when the control or feedback signal is wrong. When we look at the automaton model of the system, where each control signal and its feedback is represented as a transition from one state to another, it gets clear that by following the transitions that are triggered in normal operating conditions, one can easily detect the abnormal behavior. Since the probability of triggering a non-existing transition is zero, the appearance of every transition-triggering symbol in any state in which that particular symbol should not occur, signals an anomaly.
This can be illustrated on the example of a heater that heats a raw material in some container. After the temperature reaches the predefined threshold, the following control signal (symbol) should be 'heater off'. The *control sequence anomaly* is detected if instead of that symbol, a symbol 'empty container' occurs, which

opens the container and releases the material. Figure 7.2 illustrate this scenario, where the anomaly is represented as a non-existing path through the automaton, i.e. $PA_{automaton} \neq PA_{observed}$.

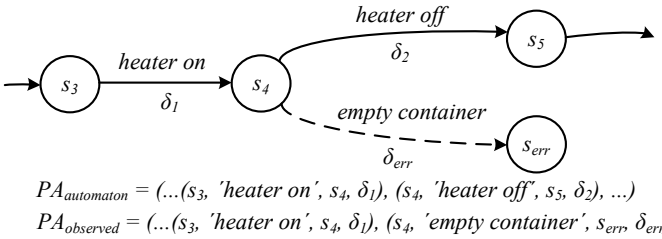


Fig. 7.2 Example of the *control sequence anomaly*.

Timing anomaly: Using the automaton behavior model, observed timings of all occurring symbols (changes in discrete control signals) are simply compared with the learned timing constraints (i.e. time intervals δ, see Definition 13). The *timing anomaly* is detected when the existing symbol is observed before or after the corresponding time interval of the transition it triggers. Based on the existing time delays in the system and the requirements imposed on time (such as real-time requirements), the *timing anomaly* can be reported to the operator in various ways. When the symbol timing is "very close" to the transition time interval (but still outside of that interval, e.g. less than a second too early or too late), some sort of warning could be signaled. When the symbol timing is however "far away" from the interval, an alarm could be raised to the operator.
An example of such an anomaly is illustrated in Figure 7.3. While being in a state s_1, an observation: $o = (a, t, \mathbf{u}, y)$ is recorded, where $t = 7$. Although the symbol is correct, it can be seen that its timing $t \notin \delta$ since it occurred two time units too late. Thus, the anomaly is detected. Time units can be expressed as any time instance (e.g. seconds or minutes), depending on the concrete system in question.

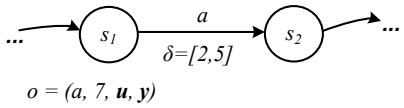
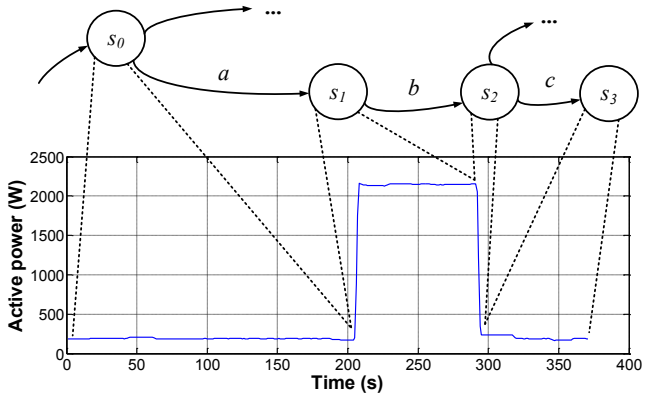


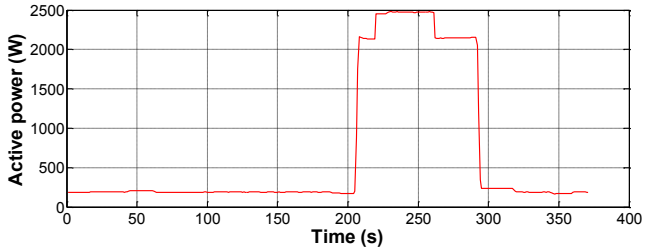
Fig. 7.3 Example of the *timing anomaly*.

Process variable anomaly: Both aforementioned anomaly types originate in the discrete control system. A very important anomaly type that can occur in the continuous physical system is the *process variable anomaly*. In every system, it is important to monitor the values of the continuous output variables that depend on the values of the continuous input variables. Typically, all such outputs need to be compared with their reference (predicted) values, which describe their normal behavior. Normally, the thresholds are given for all output variables and when the difference between the observed and the reference value exceeds the corresponding threshold, an anomaly is signaled. The use of learned behavior

models, which also model the behavior of continuous output variables, enables calculation of reference values dynamically, based on provided inputs.
A real-world example of the *process variable anomaly* is given in Figure 7.4 (that we originally published in [FFP+12]). A one-clock stochastic deterministic hybrid automaton behavior model has been learned for a component of a hybrid system. The monitored output process variable is the component active power, whose normal behavior (prediction of the model) is given in Figure 7.4(a). This figure also shows several states of the corresponding learned automaton, whose θ functions approximate marked segments of the signal. Figure 7.4(b) shows the active power signal observed during the runtime of the system. It can be seen that the *process variable anomaly* has been detected in the automaton state s_1.



(a) Predicted behavior (normal).



(b) Observed behavior (energy overconsumption).

Fig. 7.4 Real-world example of the anomalous energy consumption [FFP+12].

7.3 The ANODA Algorithm

In order to demonstrate the usability of learned 1-SDHA models in anomaly detection applications, we have developed the *ANOmaly Detection Algorithm* (ANODA) [VKBNM11b]. It is presented in this section.

The ANODA algorithm runs the model in parallel with the system and detects all significant discrepancies between their behaviors, which could indicate: *control*

sequence, *timing*, and *process variable* anomalies that are explained in the previous section. More specifically, the algorithm targets the following fault types:

1. unknown control event occurred (f_1),
2. control event occurred too early (f_2),
3. control event occurred too late (f_3),
4. signal zero value (f_4),
5. signal drop (negative offset into the signal, f_5),
6. signal jump (positive offset into the signal, f_6).

Fault f_1 is a *control sequence anomaly*, f_2 and f_3 are *timing anomalies*, while faults f_4, f_5, and f_6 represent *process variable anomalies*.

Since it works in an online manner, the ANODA algorithm receives the observations of a system periodically, at the rate that corresponds to the sampling rate of the system. The sampling rate could vary significantly based on the application area (e.g. 1 sample per second (1 Hz) for fast processes or 1 sample per minute for processes with large time delays).

The ANODA algorithm is given in Figure 7.5. It uses the learned one-clock stochastic deterministic hybrid automaton (1-SDHA) A for monitoring the behavior of a running system. Its observations, which include the observed symbol $a \in \Sigma$, its timing t, the vector of continuous input values $\boldsymbol{u}$, and the observed continuous output value y are periodically provided to the algorithm. In addition, the ANODA uses case-based expert knowledge, which includes a predefined alarm threshold ξ and the measurement range $[LR, HR]$ of the monitored output y. The difference between y and the predicted output value y_p is calculated and compared with the deviation ξ from the measurement range. In case this difference is lower or higher than the allowed value, an alarm is triggered.

The algorithm works as follows. First for every received observation $o = (a, t, \boldsymbol{u}, y)$, the set T' is defined that contains all transitions from the current state s_{curr} that can be triggered by the symbol a (line 1 in Figure 7.5). The minimum t_{min} and the maximum t_{max} timing of the appearance of the symbol a is found (if $T' = \emptyset$ these values are not defined and will not be further used by the algorithm). In case T' is an empty set, the fault *unknown event* is detected (lines 2-3). Otherwise, the transitions $\tau \in T'$ are iterated (line 5), and it is checked if any of them has the time interval δ_τ that contains the observed symbol time t (line 6). If none such transition exists, it is checked how much earlier or later than expected has the symbol a occurred, by comparing its timing t with t_{min} and t_{max} values respectively. Based on this comparison, either the fault *event too early* (lines 22-23) or the fault *event too late* (lines 24-25) is signaled to the operator. However, if the transition τ is found whose timing constraint δ_τ contains t and whose destination is some state s_{new}, this state is set to be the current state s_{curr} (line 7). Based on the timing t and the inputs $\boldsymbol{u}$, the predicted output value y_p is calculated by the function $\theta_{s_{curr}}$ (line 8). Then the difference *diff* between y_p and y is calculated (line 9). In case $y_p \neq 0$ and $y = 0$, the fault *zero value of the signal* (lines 10-11) is signaled. Otherwise, based on the comparison of *diff* with the allowed deviation ξ of the monitored variable from the lower LR and upper HR value of the measurement range, the faults *signal drop* (lines 12-13), and *signal jump* (lines 14-15) can be detected respectively. If *diff* is actually within the predefined tolerance, the algorithm returns "OK" (line 17) indicating that the given observation is anomaly-free. Please note that in order for the ANODA algorithm to work, a measurement range $[LR, HR]$ needs to be known,

Given:
(1) 1-SDHA $A = (S, s_0, \Sigma, T, \Delta, P, x, \Theta)$ according to Definition 14
(2) An observation $o = (a, t, \boldsymbol{u}, y)$ (according to Definition 25).
ANODA algorithm receives such observations periodically.
(3) ξ is a predefined alarm threshold for monitoring continuous output signal (e.g. 5% or 10%)
(4) $[LR, HR]$, $LR > 0$ is a measurement range for monitored output variable y.
This range us used as a reference for discrepancy calculation.
(5) $s_{curr} := s_0$, at the beginning, the current state is the initial state
Result: detected fault (if there exists one), otherwise "OK"

```
(1)  T' = {τ = (s_curr, a, s_new, δ_τ) ∈ T}, s_curr ∈ S, a ∈ Σ, ∀s_new ∈ S, ∀δ_τ ∈ Δ
     T' ⊆ T is a set of all transitions with the source state s_curr and the symbol a.
     Find t_min = min_τ(t_1τ) and t_max = max_τ(t_2τ) where δ_τ = [t_1τ, t_2τ], τ ∈ T'
(2)  if T' = ∅ then
(3)    return fault: unknown event
(4)  else
(5)    while notEmpty(T') do
(6)      if t ∈ δ_τ then
(7)        s_curr := s_new
(8)        y_p = θ_{s_curr}(t, u)
(9)        diff = y_p − y
(10)       if y = 0 and y_p ≠ 0 then
(11)         return fault: zero value of the signal
(12)       else if diff ≥ ξ · LR then
(13)         return fault: signal drop
(14)       else if diff ≤ −ξ · HR then
(15)         return fault: signal jump
(16)       else
(17)         return OK
(18)       end if
(19)     else T' = T'\{τ}
(20)     end if
(21)   end while
(22)   if t < t_min then
(23)     return fault: event too early at least t_min − t time instances
(24)   else if t > t_max then
(25)     return fault: event too late at least t − t_max time instances
(26)   end if
(27) end if
```

Fig. 7.5 The ANODA algorithm.

i.e. it needs to be provided by the domain experts for every monitored continuous output variable.

7.4 Real-Time Properties of the ANODA Algorithm

Since the ANODA algorithm works in an online manner, it is important to analyze its real-time properties. In this section, we first cite several important definitions of real-time systems. Then we give and prove the condition under which the anomaly detection system that uses the ANODA algorithm is a real-time system.

Hybrid production systems belong to a group of systems in which data are being processed in periodic and timely manner. One of the main properties of such systems

is that there is a certain delay between reception of the input data and appearance of the output data (i.e. a processing result). This delay is called the response time of the system, and is defined in the following.

Definition 27 (The Response Time of the System [Lap04]). The time between the presentation of a set of inputs to a system (stimulus) and the realization of the required behavior (response), including the availability of all associated outputs, is called the response time of the system.

The system that has limits imposed on its allowed response time is generally known as the *Real-Time System* (RTS). One of the broadest and oldest definitions is given by Young:

Definition 28 (Real-Time System [You82]). A Real-time system is any information processing activity or system which has to respond to externally generated input stimuli within a finite and specifiable delay.

From the philosophical point of view, any practical real-world system is actually the real-time system. Therefore, a distinction is made between so-called *soft*, *hard* and *firm* RTSs. These are defined as follows:

Definition 29 (Soft Real-Time System [Lap04]). A soft real-time system is one in which performance is degraded but not destroyed by failure to meet response-time constraints.

Definition 30 (Hard Real-Time System [Lap04]). A hard real-time system is one in which failure to meet a single deadline may lead to complete and catastrophic system failure.

Definition 31 (Firm Real-Time System [Lap04]). A firm real-time system is one in which a few missed deadlines will not lead to total failure, but missing more than a few may lead to complete and catastrophic system failure.

The examples of a soft, hard and firm RTS are respectively an automated teller machine, a flight control system of a combat aircraft, and an operating system such as UNIX that responds to user commands within several seconds [Lap04]. Further RTS applications, such as process control and manufacturing, can be found in [BW01].

A failure of the anomaly detection systems in production facilities to timely detect emerging anomalies and to trigger the appropriate alarms correspondingly, could result in catastrophic failures that jeopardize human lives and property. Such systems need to be sufficiently fast, accurate and reliable. Therefore, they represent a type of hard RTS. Since they need to monitor the condition of a system in an online manner (i.e. during runtime), they periodically receive a set of inputs and need to periodically return the predictions of the outputs. We now give and prove the following theorem that gives the condition under which the anomaly detection system, which employs the ANODA algorithm, is a hard RTS:

Theorem 15. *Anomaly detection system that uses the ANODA algorithm for online monitoring of a monitored system is the RTS if it holds:*

$$T_{resp} \leq \frac{1}{f_s}, \tag{7.1}$$

where T_{resp} is the response period of the ANODA algorithm, and f_s is the sampling rate of a monitored system. The rate f_s corresponds to the rate with which the algorithm receives observations from the system.

Proof. We use the proof by contradiction. Let us assume the following:

$$T_{resp} > \frac{1}{f_s}. \tag{7.2}$$

As already explained, the ANODA algorithm periodically receives the observations of a system (see Definition 25) and checks the correctness of the observed control signals and their timings, as well as the normality of the values of continuous output variables. It uses the values of continuous input variables to predict the values of continuous outputs. Then it compares these predictions with real, observed values of the outputs. If their discrepancy is larger than some predefined threshold, the algorithm signals an appropriate anomaly.

Let $t_{o_1}, t_{o_2}, t_{o_3}...$ be the absolute timings of the corresponding observations $o_1, o_2, o_3...$ that the ANODA algorithm periodically receives with the sampling rate f_s. Let $t_{r_1}, t_{r_2}, t_{r_3}...$ be the absolute timings of the corresponding responses of the algorithm. These timings are denoted in Figure 7.6, which shows the communication between the monitored and the anomaly detection system.

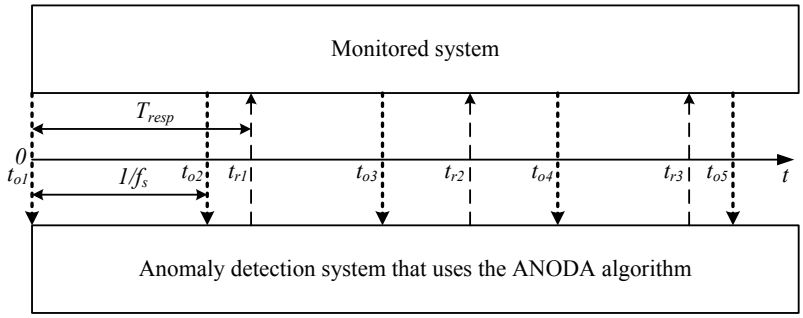


Fig. 7.6 Timings of observations and responses for $T_{resp} > \frac{1}{f_s}$.

In general case, there is no parallel processing, i.e. the ANODA algorithm can process only one observation at the time. For example, the processing of the observation o_2 can only start at the time t_{r_1}, i.e. after the processing of the observation o_1 has been finished. From Definition 28 it follows that the difference $t_{r_i} - t_{o_i}, \forall i$ needs to be finite, in order for a system to be a RTS.
From (7.2) we further have:

$$T_{resp} - \frac{1}{f_s} = \Delta t. \tag{7.3}$$

Timings of responses of the ANODA algorithm are then (as shown in Figure 7.6):

$$t_{r_1} = T_{resp},$$
$$t_{r_2} = 2 \cdot T_{resp},$$

$$\vdots$$
$$t_{r_i} = i \cdot T_{resp},$$
$$\vdots$$

Timings of the observations are:

$$t_{o_1} = 0,$$
$$t_{o_2} = \frac{1}{f_s},$$
$$\vdots$$
$$t_{o_i} = (i-1) \cdot \frac{1}{f_s},$$
$$\vdots$$

By subtracting the timing of the i-th observation from the timing of the i-th response it is obtained:

$$t_{r_i} - t_{o_i} = i \cdot T_{resp} - (i-1) \cdot \frac{1}{f_s} = i \cdot \left(T_{resp} - \frac{1}{f_s} \right) + \frac{1}{f_s}. \tag{7.4}$$

From (7.3) we get:

$$t_{r_i} - t_{o_i} = i \cdot \Delta t + \frac{1}{f_s}. \tag{7.5}$$

In the limit it is obtained:

$$\lim_{i \to \infty} (t_{r_i} - t_{o_i}) = \lim_{i \to \infty} \left(i \cdot \Delta t + \frac{1}{f_s} \right) = \infty, \tag{7.6}$$

which means that the response time of the ANODA algorithm gradually grows with the number of received observations and eventually gets infinite. That is contradictory to Definition 28.

In the online monitoring, it is important that the algorithm returns the feedback about the normality of the current observation, before the next observation comes. Only in that way the potentially required actions (in case the anomaly is detected) could be timely undertaken. Otherwise, a catastrophic system failure could occur. Therefore, when the condition (7.1) given in Theorem 15 is satisfied, the anomaly detection system that uses the ANODA algorithm for online monitoring is a hard RTS. □

7.5 Conclusion

This chapter gave an insight in one of the most important application areas of behavior models: the model-based anomaly detection. It introduced the means of detecting anomalous behavior of a system by using automatically learned behavior models. Our contributions given in this chapter are:

1. We presented our ANODA algorithm, which uses automatically learned behavior models for detecting abnormal behavior of a hybrid production system. Models are learned using the HyBUTLA algorithm presented in the previous chapter.
2. We have proven that the anomaly detection system, which uses the ANODA algorithm, satisfies the property of the real-time system under the given condition (Theorem 15). When this condition is satisfied, the algorithm can be applied in the online monitoring of a system behavior.
3. In addition to fulfilling the real-time property condition and having the behavior model available, the ANODA algorithm requires the following expert knowledge:
   - sampling rate f_s of a monitored system, which is typically known for every plant (needed for evaluating the real-time condition),
   - response period T_{resp} of the ANODA algorithm, which needs to be obtained separately from simulations for every concrete system (needed for evaluating the real-time condition),
   - measurement range $[LR, HR], LR > 0$ for every process variable that we want to monitor (needed for detecting process variable anomalies),
   - a predefined alarm threshold ξ that defines the allowed deviation of a monitored variable from its normal value.

Part IV

Case Studies in Learning and Anomaly Detection

Chapter 8

Real-World Plants

In this chapter, we demonstrate the practical applicability of our HyBUTLA and ANODA algorithms in the real-world. We conducted experiments in two running production facilities, i.e. in the plant of Jowat AG in Detmold, Germany, and in the Lemgo Model Factory at the Institute Industrial IT in Lemgo, Germany. Our main contributions given in this chapter are the following:

- we give the comparative empirical analysis on learning several types of stochastic finite automata, including 1-SDHAs,
- we identify the important trade-off between the model size reduction and the accuracy of approximating the continuous dynamics in a system, which enables selection of the appropriate model depending on the application area,
- in the real plant, we show that the prediction errors of the HyBUTLA algorithm drop as more data are used for learning,
- we show how even behavior models identified using only 12 learning examples can detect six types of faults (described in Section 7.3) in the real system with excellent or acceptable accuracies.

The chapter is structured as follows. First, in Section 8.1 we define several criteria for comparing the four algorithms that learn different stochastic finite automata, namely the algorithms ALERGIA, MDI (both for learning SDFAs), BUTLA (for learning 1-SDTAs) and HyBUTLA (for learning 1-SDHAs). We have evaluated and compared them using a dataset coming from the plant of Jowat AG. Section 8.2 describes the Lemgo Model Factory and presents several models learned using the HyBUTLA algorithm for one of its components. We devoted a special attention to applying these models to anomaly detection using the ANODA algorithm in Section 8.3. Finally, the chapter is concluded in Secion 8.4.

While some experimental results presented in this chapter are new, some are already published with various experimental settings and parameters in [VKBNM11a, VKBNM11b, NSV+12, FFP+12, Vod12, Vod13, VMN13].

8.1 Comparative Empirical Analysis on Learning Automata

In this section, we present the evaluation results of the ALERGIA, MDI, BUTLA, and HyBUTLA algorithms (see Section 2.4.2) using the same data coming from the plant of Jowat AG.

In general, different algorithms are often applied in different areas using different datasets. Therefore, it is often hard to compare them empirically, which creates a lack of comparative analyses in many fields. Such is the field of grammatical inference and learning stochastic finite automata, where algorithms learn somewhat specific structures for which common comparison criteria have to be defined [VMN13]. In order to help solving this issue, we conducted the empirical analysis given in this section. Since the HyBUTLA is to date the only algorithm capable of learning hybrid automata, we cannot compare it with other hybrid automata learning approaches. However, by ignoring the continuous data and the timing information, we were able to compare the models it learns with the models learned by the other aforementioned three algorithms. Moreover, the HyBUTLA is the only algorithm out of the four algorithms that includes the splitting step. To make the comparison possible, the splitting step (i.e. the *split* function given in Subsection 6.5.2) was here excluded from consideration. The common comparison criteria are described in the following subsection.

8.1.1 Common Criteria for Evaluating Learning Algorithms

First, we have defined the common criteria for evaluating the learning algorithms. We have originally published them in [VMN13] and they are given as follows:

#states: The number of states is the primary measure of the automaton size. In general, the goal is to obtain the smallest possible behavior model of a system, with the highest possible accuracy of representing system dynamics.

#merges: The number of successful merges tells how many pairs of states have been merged during learning. It is closely related to the number of states, as the sum: (*#states* + *#merges*) equals to the number of states in the prefix tree. State merging is used to reduce the automaton size, but also to increase its generality. Intuitively, the more successful merges, the higher is the generalization ability of the algorithm.

#comparisons: During a merging step, a search procedure is performed in order to find as many compatible states as possible. The more comparisons are made, the higher is the chance that compatible states will be found and merged.

#determinizations: As stated earlier, some algorithms use a top-down, while others use a bottom-up merging order. While merging in a top-down order, the large subtrees of a prefix tree are encountered. Thus, the occurrence of non-determinism in the automaton is more frequent. The number of determinizations indicates the portion of non-determinism created during the merging step, which needs to be resolved by the learning algorithm.

Size reduction (%): Size reduction is the measure of the relative difference between the sizes of the prefix tree (*#PTAstates*) and the final automaton (*#states*). It is calculated as (*#PTAstates - #states*) $\cdot 100$ / *#PTAstates*. Successful algorithms can achieve high rates of size reduction.

R^2 *(%)*: The averaged coefficient of determination R^2 *(%)* over the automaton states (given by the expression (6.12)) shows the portion of variability in the continuous data that is accounted for by the regression function used for approximation. Average R^2 can be measured only for the HyBUTLA algorithm and it shows its ability to model the continuous dynamics of the system. In the following experiments, *Multiple Linear Regression with Linear Terms* (MLR-LT) was used as the regression method [HTF08].

8.1.2 Algorithm Evaluation at Jowat AG

With the courtesy of the Jowat AG company, we were able to conduct experiments at one of the plants, using the ALERGIA, MDI, BUTLA, and HyBUTLA algorithms. We have originally published the following two paragraphs in [VMN13] (the first one being publicly available description of the company).

The Jowat AG with headquarters in Detmold is one of the leading suppliers of industrial adhesives. These are mainly used in woodworking and furniture manufacture, in the paper and packaging industry, the textile industry, the graphic arts, and the automotive industry. The company was founded in 1919 and has manufacturing sites in Germany in Detmold and Zeitz, plus three other producing subsidiaries, the Jowat Corporation in the USA, the Jowat Swiss AG, and the Jowat Manufacturing in Malaysia. The supplier of all adhesive groups is manufacturing approx. 70,000 tons of adhesives per year, with around 790 employees. A global sales structure with 16 Jowat sales organizations plus partner companies is guaranteeing local service with close customer contact.

The data was logged in one of the plants, during production of one product. In total 14 production cycles were logged. The modeled part of the system is the input raw material subsystem, which contains 6 material supply units (smaller containers) connected to a large container where materials are mixed. Recorded discrete variables are 15 valve open signals and their feedbacks (in total 30 discrete variables). The continuous output variable whose dynamics was learned is the large container weight. Continuous input variables are weights of 6 smaller containers and the pressure of the raw material pump. The results of the algorithms' comparison are given in Table 8.1.

Table 8.1 Algorithm comparison for Jowat AG data.

Criterion	**Algorithm**			
	ALERGIA	MDI	BUTLA	HyBUTLA
#states	27	16	17	13
#merges	418	429	473	507
#comparisons	1025	605	4526	3576
#determinizations	348	578	150	111
Size reduction (%)	93.93	96.4	96.5	97.5
MLR-LT R^2 (%)	-	-	-	89.8

We interpret these results as in [VMN13]. In modeling the component of this real-world system, all four algorithms achieved high size reduction rates. This shows

their ability to produce small and more general models. Learned models with 13 to 27 states can be easily visualized, understood and interpreted by human experts. Therefore, they provide a good insight in the system modes of operation and its behavior in general. Obtaining such insight from prefix trees with several hundreds of states (in this case 400–500 states) is not possible. It can be seen that the bottom-up algorithms (i.e. BUTLA and HyBUTLA) perform much more comparisons than the top-down algorithms (ALERGIA and MDI). Although this negatively affects their runtime, it basically means that they perform a more thorough search for compatible states in the prefix tree. Unfortunately, at the time of writing this thesis the only available implementations of the algorithms were on different platforms, so we could not make a reliable comparison of the runtime itself. Another interesting aspect is that the top-down algorithms create more non-determinism during merging, which needs to be resolved. Thus, they perform significantly more determinizations. Last but not the least, even with relatively simple regression method, such as multiple linear regression with linear terms, and without applying the additional splitting step of the HyBUTLA algorithm, it still achieved a relatively high R^2 value of around 90%. In the following, such learned behavior models will be used for anomaly detection.

8.2 Learning Behavior Models for the Lemgo Model Factory

This section first briefly describes the second real-world production facility that we used for demonstrating our approach, the so-called Lemgo Model Factory. Then, the models learned for a component of this facility using the HyBUTLA algorithm are presented. Such models are used for executing the anomaly detection experiments, which are presented in Section 8.3.

8.2.1 Plant Description

The *Lemgo Model Factory* (LMF) is the exemplary hybrid production system at the Institute Industrial IT in Lemgo, Germany. It represents a small plant for storing, transporting, processing and packing bulk materials, such as corn. It is made of several modules, namely: the storage system, transportation system, weighting station, bottle-filling mechanism, material-processing facility, product packing system, bearing robot, and lid robot. These modules consist of a number of components, such as distributed PLCs, industrial networks (using e.g. the PROFINET protocol [PM08]), conveyor belts, and a popping machine. These components are already identified by the manually created parallelism model, which is explained in Subsection 2.1.1. The plant comprises around 250 measurable discrete and continuous signals. We show one part of the plant's interior in Figure 8.1.

The component of the LMF that we modeled using our HyBUTLA algorithm is the popping machine. It consists of one container, the fan and the heater. First, the corn is delivered to the container and then the heater and fan are activated. The fan blows the hot air inside the container and creates popcorn as a result.

Logs made during described production cycle represent one learning example of the popping machine. They comprise the time stamp and the logs of: six discrete binary control signals, five continuous input signals and one continuous output signal.

Fig. 8.1 One part of the Lemgo Model Factory.

This signal is the active power of the popping machine heater. Figure 8.2 shows its typical time diagram. The normal measurement range of this signal is 150–3250 W. This information (expert knowledge) is important for the ANODA algorithm, which can detect significant deviations of this signal from the lower or upper bound of the measurement range. The data sampling rate at the popping machine is 1 Hz. For learning models, we had logs of 12 production cycles at our disposal, which are by experts identified as normal (i.e. recorded during normal operating conditions in the plant). Logs of the additional 13th cycle are used as a test example for the anomaly detection experiments given in the following section. In the average, the example length is 191 samples (i.e. the average production cycle lasts around three minutes).

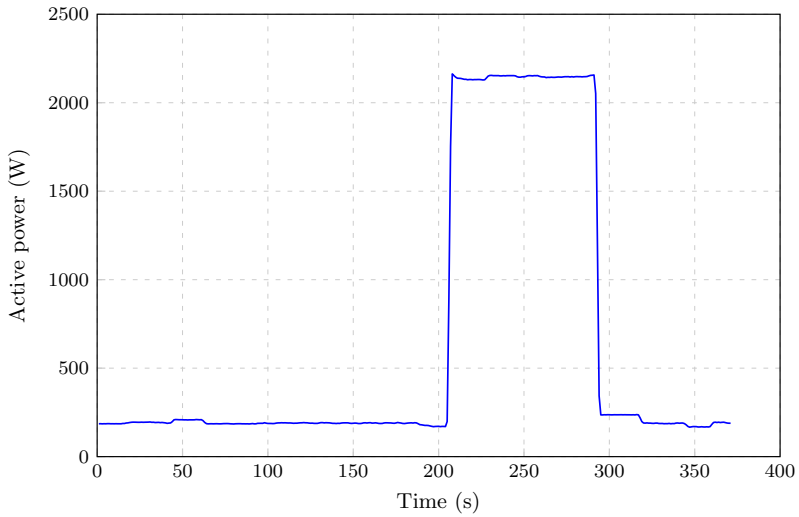


Fig. 8.2 Active power signal of the popping machine heater.

8.2.2 Models Learned with the HyBUTLA Algorithm

In our papers cited at the beginning of this chapter, we have published several case studies conducted at LMF and gave properties of various models learned using the HyBUTLA algorithm. In all those cases, an interesting trade-off was observed that we want to present and explain here. To that aim, we first give properties and learning times of several popping machine models identified by our algorithm. The general goal was to learn the smallest possible model, with the highest possible accuracy of approximating the continuous output signal.

The properties of the *Prefix Tree Acceptor* (PTA) and two specific learned behavior models are given in Table 8.2. We denoted the first learned model by A_{min}. It is the smallest model identified by the HyBUTLA algorithm without the application of the additional splitting step (model obtained for $\alpha = 0.25$, see the expression (6.3)). The $split$ function (see Subsection 6.5.2) was applied to this model and resulted in the second model denoted by A_{split}. For these three models (i.e. PTA, A_{min} and A_{split}) Table 8.2 gives: their number of states, the size reduction (in relation to the PTA size), the average model coefficient of determination achieved using the multiple linear regression with linear terms ($MLR - LT$ R^2), and total learning times.

Table 8.2 Properties of learned behavior models and their learning times.

Properties	Automaton		
	PTA	A_{min}	A_{split}
#states	52	7	24
Size reduction (%)	0	86.5	53.9
MLR-LT R^2 (%)	78.3	75.1	93.5
MLR-LT time (s)	3.4	7.1	185.3

It can be seen that by merging the PTA states a very small model A_{min} is obtained with only seven states. Thus, the reduction of the PTA size obtained by merging its states is over 86%. By applying the splitting step, this size reduction has unfortunately decreased, but the MLR-LT R^2 of the final A_{split} model has increased by around 18% in the absolute value comparing to A_{min}. It is interesting that this final model has higher MLR-LT R^2 even than the PTA itself.

Table 8.2 presents only three specific learned models. However, during the splitting step a number of models with different sizes and coefficients of determination were created. We have summarized their main properties and showed them graphically in Figure 8.3. Three aforementioned models are denoted at the bottom of the figure. This figure illustrates the existing trade-off between the model size reduction and the accuracy of approximating the continuous output signal. It demonstrates the benefit that our $split$ function has brought to the HyBUTLA algorithm. Depending on the application area, the flexibility now exists to select a model with the appropriate size-accuracy ratio. Furthermore, the model with the R^2 of over 93% (A_{split}) is in this example learned using the simple MLR-LT method for regression. This approach is several times faster than our previous approach without the $split$ function, where continuous output signals had to be learned with neural networks in order to achieve high R^2 values [VKBNM11a].

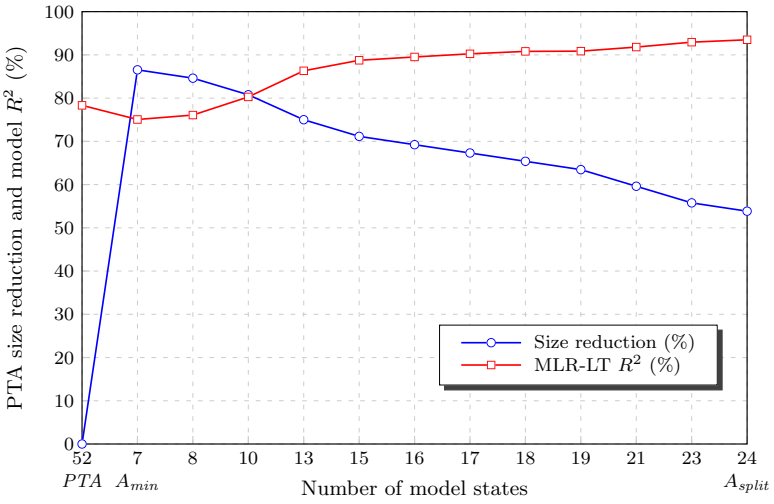


Fig. 8.3 Trade-off between the size reduction and function approximation accuracy.

8.3 Anomaly Detection Experiments

This section demonstrates the usability of behavior models learned with our Hy-BUTLA algorithm in the anomaly detection application. Experiments are conducted at the LMF using our ANODA algorithm for anomaly detection (see Section 7.3). We have already published several such studies in papers cited at the beginning of this chapter. However, they focus on special types of anomalies (e.g. signal drop by 10% [VKBNM11a, FFP$^+$12]). Here we give much more comprehensive results (e.g. signal drop by various values, ranging from 2% to 25%).

8.3.1 The Setting

As already explained in Section 7.3, the ANODA algorithm targets six types of faults, namely (i) unknown control event occurred (f_1), (ii) control event occurred too early (f_2), (iii) control event occurred too late (f_3), (iv) signal zero value (f_4), (v) signal drop (negative offset into the signal, f_5), and (vi) signal jump (positive offset into the signal, f_6). Faults f_1, f_2, and f_3 originate in discrete control system, while faults f_4, f_5, and f_6 occur in continuous physical system.

For evaluating the performance of the ANODA algorithm in detecting these faults, the ratios of the numbers of correctly and incorrectly classified samples are used. These are defined by a table called *confusion matrix* [Faw06]. The general confusion matrix is shown in Table 8.3 and it presents the number of *True Positive*, *False Positive*, *False Negative*, and *True Negative* samples. These numbers are explained as follows:

- *True Positive* (TP) is a sample that is truly anomalous and recognized as such by the anomaly detection algorithm.
- *False Positive* (FP) is a sample that is anomaly-free, but recognized as anomalous.

- *False Negative* (FN) samples contain an anomaly, which is not detected by the algorithm.
- *True Negative* (TN) is a truly normal sample that is recognized as such by the algorithm.

Table 8.3 The confusion matrix.

Test outcome	**Actual condition**	
	Positive	Negative
Positive	*True Positive* (TP)	*False Positive* (FP)
Negative	*False Negative* (FN)	*True Negative* (TN)

Based on the numbers of these samples, the following performance metrics are calculated (all expressed as a percentage):

- *Specificity* (true negative rate) is the probability that the test outcome is negative (i.e. anomaly-free), given that the condition is indeed negative (not anomalous). We calculate it using the formula:

$$Specificity = \frac{\sum TN}{\sum FP + \sum TN}. \tag{8.1}$$

- *Sensitivity* (true positive rate) is the probability that the test outcome is positive (i.e. the anomaly is detected), given that the condition is indeed positive (anomalous). It is calculated as:

$$Sensitivity = \frac{\sum TP}{\sum TP + \sum FN}. \tag{8.2}$$

- *Accuracy* is a single number that summarizes capabilities of the anomaly detection algorithm. It represents the probability of accurate test outcomes:

$$Accuracy = \frac{\sum TP + \sum TN}{\sum TP + \sum TN + \sum FP + \sum FN}. \tag{8.3}$$

We use these performance metrics to evaluate how good our ANODA algorithm detects certain deviations of monitored signals from their normal values.

As explained before, the ANODA algorithm receives observations (measurements of input and output signals) periodically, uses inputs and learned behavior model to predict outputs, and compares those predictions with the outputs' real values (as shown in Figure 7.1). In our real-world example at the LMF, the sampling rate is 1 Hz, which means that the algorithm receives one observation from the system per second. Following Theorem 15, the ANODA algorithm can work only when its response time is not greater than the period of receiving observations from the system. Our analysis and simulations have shown that for the popping machine component of the LMF, algorithm's maximum response time is 1.4 ms. Therefore, the condition given by Theorem 15 is in this case fulfilled.

Please note that in models used for anomaly detection in experiments given here, the behavior of continuous output signals (i.e. the θ functions of automata) is learned using relatively simple multiple linear regression with linear terms. The usage of

more sophisticated regression methods (e.g. support vector regression) would most likely improve the results at the cost of an increased runtime of our approach.

8.3.2 Anomalies in the Discrete Control System

8.3.2.1 Anomaly Detection

The faults *unknown control event occurred*, *control event occurred too early* and *control event occurred too late* originate in the discrete control system. In order to evaluate how good the ANODA algorithm detects such faults in the popping machine component of the LMF, we conducted a number of experiments.

The behavior model (one-clock stochastic deterministic hybrid automaton) was learned using logs of 12 production cycles (i.e. *the training set*). The aforementioned faults are inserted in the logs of the 13th cycle, which was not used for learning (i.e. *the test set*). For each fault type, we performed 100 experiments whose results are averaged. In the absence of these artificially induced faults, the test cycle fully complies with the learned model, i.e. normally no alarms would be signaled. Therefore and due to the fact that all anomalies in the discrete control system can be detected in a deterministic way as described in Section 7.2, maximum values of specificity, sensitivity and accuracy are reached.

8.3.2.2 Generalization of Learned Models

Aforementioned positive result tells however nothing about the generalization of learned models, i.e. it is questionable how good a learned model generalizes to an independent dataset that was not used for learning. We want to show that the number of wrongly detected faults by the learned model and the ANODA algorithm is significantly smaller when more examples are used for learning. In order to assess the model generalization, we use a technique called the *K-fold cross validation* [RN10]. The available dataset is divided into K equal subsets of learning examples. Then K rounds of learning are carried out, each time keeping $1/K$ of examples as the test set, while remaining are used for learning. For every learned model, an error is measured on test examples and the results are averaged. In our experiments on generalization, we first used the modified 2-fold cross validation. Modification comes from that fact that the number of examples available for these experiments from the popping machine component of the LMF is 13, and therefore two subsets cannot have the same size.

Nevertheless, we first picked seven cycles as a training set and used other six for testing. Training cycles are selected using a random number generator with an uniform distribution. The results are shown in Table 8.4. Since both the training and test set comprise the positive (i.e. normal) learning examples, we have measured the number of wrongly detected faults in the discrete control system for each individual test cycle and then summarized the results. As Table 8.4 shows, only *unknown event* fault had low prediction error of about 5%, while both *event too early* and *event too late* had unacceptable errors of around 13% and 29% respectively. In total, around 47% of correct events were falsely recognized as anomalous, since they are not represented in the learned model of normal behavior.

Table 8.4 Results of the modified 2-fold cross validation for the component of the LMF. Cycles: 2, 3, 6, 7, 10, 12, and 13 are used for learning.

	Wrongly detected faults				
Test cycle ID	#Unknown events	#Events too early	#Events too late	#Incorrect events	#Total events
1	1	0	3	4	4
4	0	1	1	2	8
5	0	2	3	5	8
8	0	1	1	2	7
9	0	1	3	4	7
11	1	0	0	1	4
#False alarms	2	5	11	18	-
Prediction error (%)	5.26	13.16	28.95	47.37	-

After folding the training and test set in our modified 2-fold cross validation, a new model was learned (this time using six cycles), and the prediction error was measured on the remaining seven (in this case test) cycles. Results are shown in Table 8.5. In this case, the *event too late* fault had a low prediction error of about 5%, while the prediction error of the other two considered faults was unacceptably high. In this case, around 36% of the total number of correct events are wrongly recognized as incorrect.

Table 8.5 Results of the modified 2-fold cross validation for the component of the LMF. Cycles: 1, 4, 5, 8, 9, and 11 are used for learning.

	Wrongly detected faults				
Test cycle ID	#Unknown events	#Events too early	#Events too late	#Incorrect events	#Total events
2	1	2	1	4	4
3	1	0	0	1	1
6	1	0	0	1	1
7	1	0	0	1	5
10	0	2	0	2	8
12	0	1	0	1	8
13	1	1	1	3	9
#False alarms	5	6	2	13	-
Prediction error (%)	13.89	16.67	5.56	36.11	-

The prediction errors from Table 8.4 and Table 8.5 are averaged for each fault and given in Table 8.6.

Table 8.6 Averaged prediction error from Table 8.4 and Table 8.5.

	Wrongly detected faults			
	#Unknown events	#Events too early	#Events too late	#Incorrect events
Average prediction errors (%)	9.58	14.92	17.26	41.74

In order to observe the quality of learned models with respect to the increased size of the training set, we also conducted validation experiments using the *Leave-One-Out Cross Validation* (LOOCV) technique, which is a type of the *K-fold cross validation*. The LOOCV is an extreme case where the number K equals the number of the available learning examples. It is the most computationally expensive version of the K-fold cross validation. Table 8.7 gives the results of the LOOCV for the learned behavior model of the LMF popping machine. It can be seen that, comparing to Table 8.6, significantly lower prediction errors for all three fault types are observed. The total number of the events wrongly recognized as incorrect has dropped by about 19% and 7% in the absoulute values when 12 examples instead of respectively seven and six are used for learning the model (i.e. *#Incorrect events* from Table 8.7 is compared with *#Incorrect events* given in Table 8.4 and Table 8.5, respectively). In general, these experiments showed the expected result - lower prediction errors are obtained when models are learned using larger training datasets.

Table 8.7 Results of the LOOCV for the component of the LMF.

	Wrongly detected faults				
Test cycle ID	#Unknown events	#Events too early	#Events too late	#Incorrect events	#Total events
1	0	1	3	4	6
2	1	0	2	3	4
3	1	0	3	4	5
4	0	1	1	2	8
5	0	0	1	1	8
6	0	2	1	3	11
7	1	0	0	1	5
8	0	0	0	0	7
9	0	1	1	2	7
10	0	1	0	1	8
11	0	1	2	3	8
12	0	0	0	0	8
13	1	1	1	3	9
#False alarms	4	8	15	27	-
Prediction error (%)	4.26	8.51	15.96	28.72	-

8.3.3 Anomalies in the Continuous Physical System

Here we present the capabilities of the ANODA algorithm to detect anomalies in the continuous physical system of the LMF popping machine component. Targeted faults are *signal zero value*, *signal drop* and *signal jump*. They occur in the continuous output signal of this component, i.e. in its active power signal.

8.3.3.1 Signal Zero Value

The fault *signal zero value* is detected when the model prediction of the value of continuous output variable is different from zero, while its corresponding observed

value equals to zero. Detection of this fault could for example indicate the sensor failure or interrupted communication cable in the system. In order to evaluate the capability of the ANODA algorithm to detect this particular fault, a number of experiments were conducted.

The used behavior model is learned from the training set of 12 logged LMF production cycles. The remaining cycle was used for testing in a way that a number of non-zero values of its output variable were artificially set to zero. Experiments are conducted for three portions of anomalous samples in the cycle, namely for 30%, 50% and 70% of them. This means that at first, we have artificially set the value of continuous output variable in 30% of randomly chosen samples in the test cycle to zero, and then measured the ability of the ANODA algorithm to detect those samples. Then, we repeated the experiments for 50% and 70% of the anomalous samples in the test cycle. For each of these portions of anomalous samples, the experiment was repeated 100 times, each time inserting zeros at different, randomly selected positions. To this aim, a random number generator with an uniform distribution is used. Table 8.8 shows the obtained specificity, sensitivity and accuracy averaged over 100 experiments for each portion of anomalous samples. For the active power signal of the LMF popping machine component (signal is shown in Figure 8.2), all instances of this fault can be detected.

Table 8.8 Detection of zero signal - testing data.

Performance metrics	**Portion of anomalous samples (%)**		
	30	50	70
Specificity (%)	100	100	100
Sensitivity (%)	100	100	100
Accuracy (%)	100	100	100

8.3.3.2 Signal Drop

The ANODA algorithm detects the fault *signal drop* when the observed value of the monitored signal drops below its predicted (normal) value by more than defined by some threshold. In industrial applications, this threshold is obtained as a deviation ξ (e.g. 5% or 10%) from the lower bound of the signal measurement range. In the case of the popping machine active power signal, an alarm is raised when the difference between the predicted and observed values gets equal to or greater than $\xi \cdot 150$ W (the measurement range for this signal is [150, 3250] W).

For performing these experiments, the same learned behavior model was used as in the case of the *signal zero value* fault. Logs of one production cycle (that was not used for learning) are used as a testing example, in which we artificially inserted a number of negative offsets from the normal signal value, ranging from 2% to 25%. Experiments are also done for three portions of anomalous samples, namely when the signal drop is inserted in 30%, 50% and 70% of the total number of samples in the example. For each portion of anomalous samples and for each deviation from the normal value, we conducted 100 experiments, each time measuring the performance metrics.

Figure 8.4 graphically shows the performance metrics for the case of having 30% of anomalous samples in the test cycle. Shown results are the average values and standard deviations of performance metrics over 100 experiments conducted for each denoted offset from the normal signal value[1]. It can be seen that the larger deviations from the normal value (i.e. negative offsets of 12% or larger) are more easily detected by the ANODA algorithm. When deviations are equal to or larger than 15%, all three performance metrics are over 90%. Small deviations are understandably harder to detect. In contrast to specificity, which reaches high value aleardy at the deviation of 5%, the sensitivity is especially low for small deviations and has in general larger standard deviation. When detecting the *signal drop* fault in these cases, the algorithm (i.e. the classifier) has difficulties recognizing truly anomalous samples, because the values of deviations are extremely small comparing to the measurement range (e.g. the deviation of 2% from 150 W is only 3 W). Therefore, lower sensitivity values are recorded for lower deviations from the normal signal value.

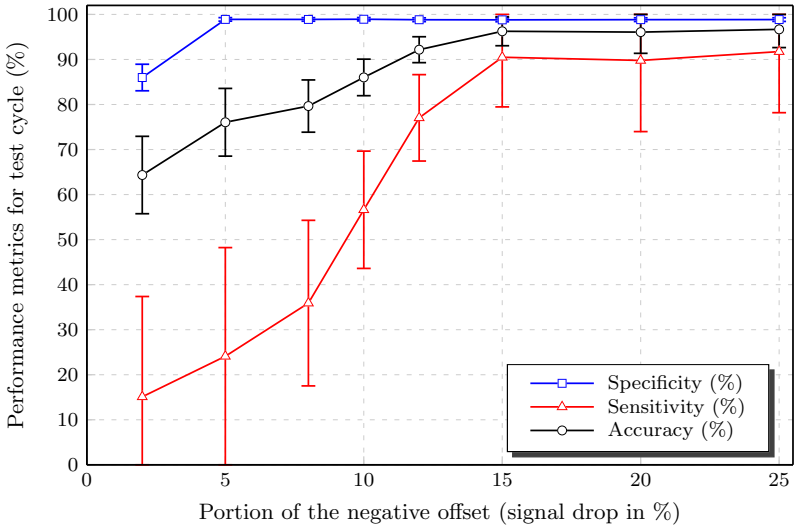


Fig. 8.4 Detection of *signal drop* for 30% of anomalous samples.

In order to validate the robustness of the ANODA algorithm in detecting the *signal drop* fault, we have additionally conducted experiments for the cases of having 50% and 70% of anomalous samples in the testing example. The goal was to evaluate if the algorithm performance will be significantly changed in the presence of the much larger number of anomalous samples. The results are graphically shown in Figure 8.5 and Figure 8.6, respectively. For all three cases (30%, 50% and 70% of anomalous samples) the trends for specificity, sensitivity and accuracy are in general very similar. It can be noted that only the accuracy is degraded in the presence of larger portions of anomalous samples for smaller deviations. This is, as mentioned above, due to the fact that the *signal drop* fault is very hard to detect for small deviations from the

[1] The results shown in figures 8.5, 8.6, 8.7, 8.8, and 8.9 also represent the average values and standard deviations of performance metrics over 100 experiments for each denoted offset from the normal signal value.

measurement range. Therefore, when more samples that are anomalous are present, the accuracy of the algorithm is somewhat lower.

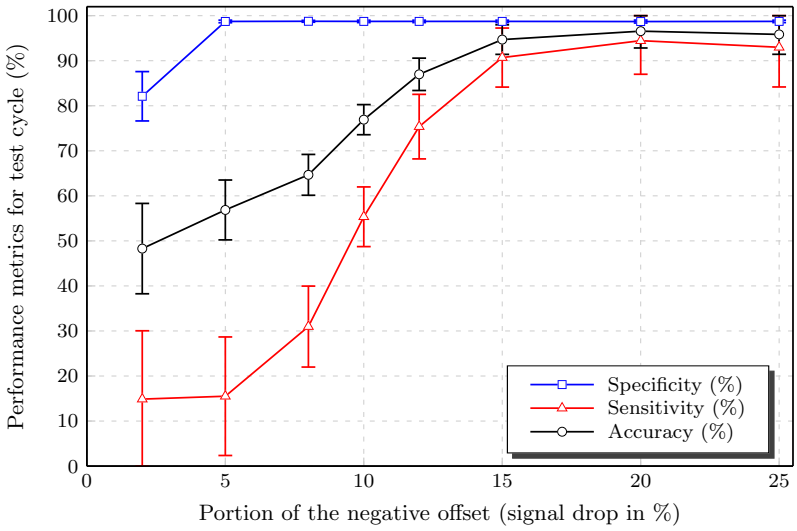


Fig. 8.5 Detection of *signal drop* for 50% of anomalous samples.

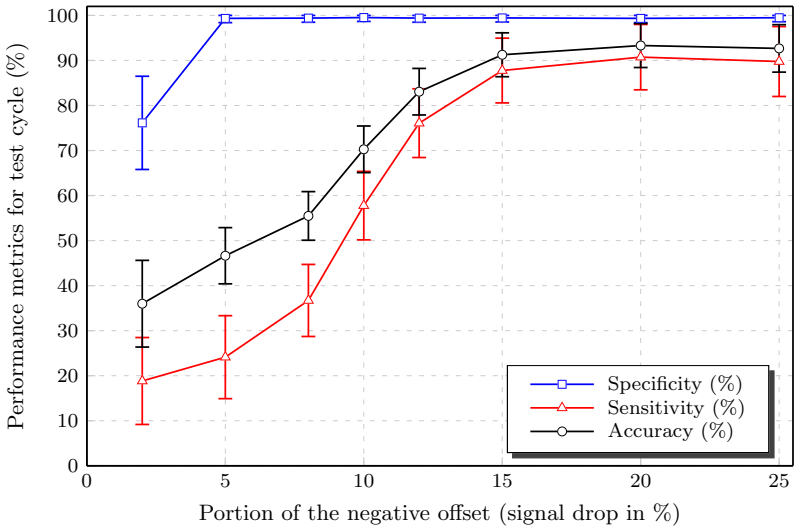


Fig. 8.6 Detection of *signal drop* for 70% of anomalous samples.

Although not as interesting as the experiments on testing data, we have also conducted the experiments on training data. First, the model was learned using 12 normal learning examples. Then, each example was modified to include a certain portion of anomalous samples (as in the experiments on test data, 30%, 50% and 70%

of anomalous samples). Such modified examples have been used as the "testing data" one at the time, each time measuring how well the model detects anomalies at corresponding positions in each particular modified training example. The performance metrics were averaged over 100 experiments performed for each negative offset and each portion of anomalous samples. For the *signal drop* fault, the obtained sensitivity is higher on training data than on aforementioned testing data, for all three portions of anomalous samples. However, the average specificity is lower on training data as well as the accuracy for larger negative offsets. In general, the standard deviations are for all three measures (sensitivity, specificity and accuracy) larger than for the testing data. The reason is most likely the averaging of the results over 12 examples, in contrast to using the real testing data with only one learning example. However, this interesting trend, as well as the lower specificity and accuracy for larger negative offsets observed on the training data, remain to be analyzed in future.

8.3.3.3 Signal Jump

The fault *signal jump* is detected when the monitored signal jumps above its predicted (normal) value by more than defined by the given threshold. Typically in the industry, the threshold for this type of fault is calculated as a deviation ξ from the upper bound of the signal measurement range. Concretely for the active power of the popping machine component at the LMF, this threshold is obtained as a product $\xi \cdot 3250$ W. When the difference between the observed and the predicted values of this signal is equal to or greater than this product, the anomaly is signaled to the operator.

The experimental setting is the same as in the case of the *signal drop* fault, i.e. the same behavior model is used, 100 experiments are performed for various deviations ranging from 2% to 25%, and experiments are repeated for 30%, 50% and 70% of anomalous samples in the test cycle.

Experimental results for the case of 30% of anomalous samples in the test cycle are given in Figure 8.7. In general, like in the case of the *signal drop* fault, better detection results are observed for larger deviations from the normal value. However, the sensitivity retains acceptable values (between 80% and 90 %) regardless of the extent of the inserted deviation. Both accuracy and specificity are below 50% for small deviations and grow to over 90% as the deviation increases over 15%. Significantly larger sensitivity for the *signal jump* fault is explained by the fact that even small deviations from the large upper bound of the active power measurement range can be detected relatively easy. This is due to their much larger absolute values. For example, the deviation of 2% from 3250 W is 65 W, which is easier to detect than the same deviation from the lower bound of the measurement range that was in this case 3 W.

For evaluating the robustness of the ANODA algorithm in detecting the *signal jump* fault, we have also conducted experiments for the cases of having 50% and 70% of anomalous samples in the test cycle. The results are given in Figure 8.8 and Figure 8.9, respectively. The general trends are similar like in the previous case. The only significant difference is that the accuracy values are slightly increased in the presence of more anomalous samples for the negative offset of 2%. As explained above, the *signal jump* fault is for the algorithm relatively easy to detect, as it works with relatively large values in this case. Nevertheless, this finding is not very interesting, since having high accuracy in detecting deviations of about 2% from the nominal value is a requirement that is rarely imposed in hybrid production systems.

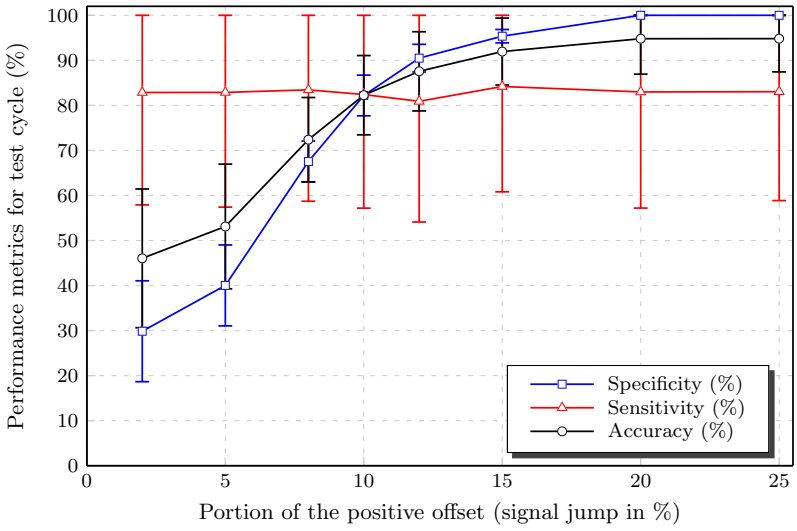


Fig. 8.7 Detection of *signal jump* for 30% of anomalous samples.

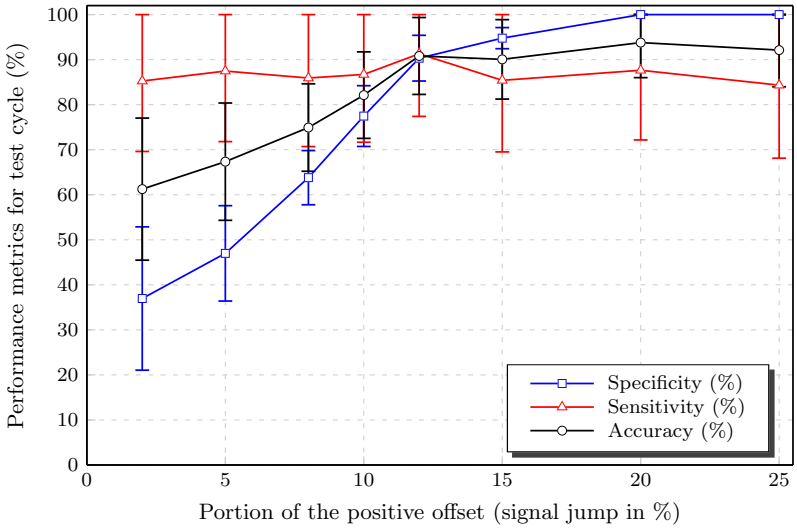


Fig. 8.8 Detection of *signal jump* for 50% of anomalous samples.

In the same way as described at the end of the previous subsection, we have conducted the experiments on training data for the fault *signal jump* as well. All 12 normal production cycles were used as the testing data one at the time, after changing the certain portion of their samples to make them anomalous. The performance metrics were in this case also averaged over 100 experiments for each positive offset and each portion of anomalous samples. For this fault, the obtained specificity is in general always higher on training data than on the testing data. Both average sensitivity and average accuracy for larger negative offsets are however lower on training data. Furthermore, the observed standard deviations are generally larger on

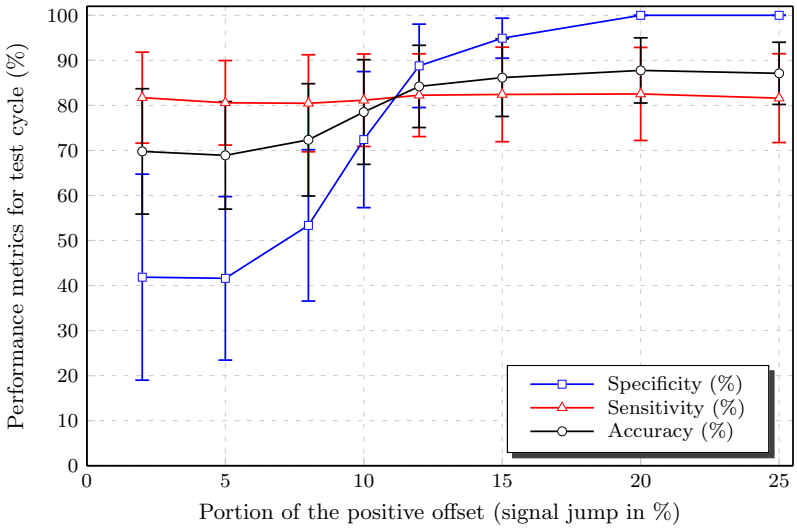


Fig. 8.9 Detection of *signal jump* for 70% of anomalous samples.

training data. We believe that this is again due to averaging the results. Nevertheless, we leave the interpretation of these results on the training data to future work.

8.4 Conclusion

In this chapter, we have demonstrated the application of our model-learning and anomaly detection approaches in the real-world. We have applied the HyBUTLA algorithm to model components of two production facilities.

At the plant of Jowat AG we have conducted the comparative empirical analysis of four algorithms for learning three types of stochastic finite automata, namely SDFA, 1-SDTA and 1-SDHA. All algorithms resulted in relatively small models (13 to 27 states) that can be easily understood, visualized and interpreted by humans. The bottom-up algorithms BUTLA and HyBUTLA make a more thorough search for compatible states than the top-down algorithms ALERGIA and MDI. This property is, however, paid with an increased runtime. Moreover, the bottom-up algorithms create much less non-determinism during the merging step.

Our general goal of learning is to obtain models as small as possible, which approximate the continuous dynamics as good as possible. On the example of learning the behavior model for a popping machine component of the Lemgo Model Factory, we have shown an interesting trade-off between these requirements. The merging step of the HyBUTLA algorithm increases the model size reduction (in relation to the size of the initial PTA model) but at the same time it decreases the accuracy of approximating the continuous dynamics of the system (i.e. its average R^2). By merging the states, it gets harder to represent the portions of continuous data that originated at those states with one regression function. The splitting step has the opposite effect. It decreases the size reduction (i.e. increases the model size) but increases the average model R^2. These two steps together enable finding an optimal

trade-off between these two properties of the model, depending on the application area.

The *K-fold cross validation*, conducted at the Lemgo Model Factory, has shown that the learned model's prediction errors in the discrete control system have significantly smaller values when more examples are used for learning. Furthermore, the anomaly detection experiments have shown that the ANODA algorithm can detect all instances of the faults: *unknown control event*, *control event too early*, *control event too late* and *signal zero value*, using the model learned by the HyBUTLA algorithm. The faults *signal drop* and *signal jump* can be detected with the acceptable performance only for larger deviations from the normal signal value (15% and larger). The experiments also showed that the increase in the portion of anomalous samples does not influence the performance of the anomaly detection system significantly. In general, all these results are promising considering that (i) only 12 learning examples are used for learning the used model and (ii) the relatively simple regression method (i.e. the multiple linear regression with linear terms) has been used to learn and later predict the behavior of the continuous part of the system.

Chapter 9

Artificial Datasets

Due to the limitations on the amount and structure of data that could be logged in a real plant, artificially generated datasets often pose the only available resource for conducting tests of certain algorithms' properties. Artificial data generation enables (i) the creation of arbitrary number of arbitrary complex learning examples and (ii) the evaluation of the learned model by comparing it to the target model that generated the data.

In this chapter, we have used artificially generated data for conducting several experiments with our HyBUTLA algorithm. The main contributions are as follows:

- we use the data generated by the benchmark target automaton to confirm our theoretical HyBUTLA convergence and polynomial runtime results experimentally,
- we conduct comprehensive experiments to show the scalability capabilities of the HyBUTLA algorithm,
- given that discrete signals in a modeled system are independent, we formally show that the maximum number of state merges, performed by the HyBUTLA algorithm, is bounded by a function that is linear in the number of those signals,
- we formally show that when the number of discrete signals in a modeled system is equal to or larger than the given limit, no state splits can be performed by the HyBUTLA algorithm,
- general observations are derived that can help practitioners in modeling hybrid systems with the HyBUTLA algorithm.

The chapter is organized as follows. In Section 9.1, we experimentally confirm the HyBUTLA convergence and polynomial runtime properties given by Theorem 14 in Section 6.6, that the class of 1-SDHAs can be identified in the limit with probability one in polynomial time. These experiments are conducted using artificial data, which are generated according to a predefined model. Section 9.2 brings the HyBUTLA scalability analysis, i.e. the learning and runtime properties of the algorithm are analyzed in the presence of the increasing numbers of discrete control and continuous physical signals in a system. We give bounds on the number of merges (Theorem 16) and number of splits (Theorem 17) that the HyBUTLA algorithm can perform under certain constraints. The chapter is concluded in Section 9.3. We have partially published both the convergence and scalability experiments in [VMN13].

9.1 Empirical Analysis of Convergence and Polynomial Runtime

In this section, we demonstrate the convergence and polynomial runtime properties of the HyBUTLA algorithm. To that aim, we use the benchmark target Reber-like model. It is based on the Reber grammar [Reb67] that has already been used as a predefined distribution for learning stochastic deterministic finite automata from data [CnCV93, CO94, CO99]. The Reber-like automaton that comprises eight states is shown in Figure 9.1. Transitions are shown with their corresponding probabilities that are the same as in [CO99], while Reber-like grammar also includes transition time intervals. For simplicity, we associated the same time intervals to all transitions and denoted the transitions that lead from different states to the same state with the same symbol. Probabilities of the two transitions whose source and destination states are the same (see transitions P(0.6) and X(0.7) in Figure 9.1), denote the ending state probabilities. For our experiments, Reber grammar was changed so that the equal subsequent symbols do not occur. This constraint is imposed by our application area (modeling technical systems), where equal subsequent symbols do not happen under normal operating conditions (e.g. a valve can open only once, and then it needs to close before the next opening).

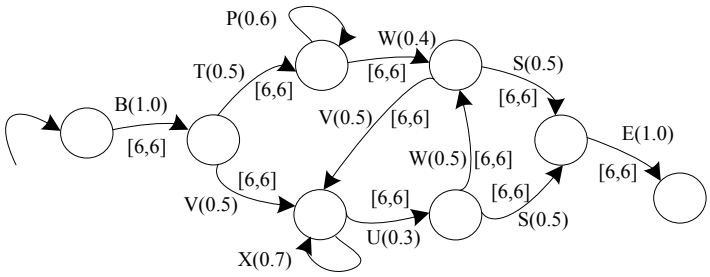


Fig. 9.1 The Reber-like model.

The generated dataset comprises 200 learning examples, drawn according to the Reber-like grammar. The average length of an example is 25 timed samples. Each dataset comprises a time stamp, three discrete variables that encode the symbols shown in Figure 9.1, one continuous input and one continuous output variable. Continuous variables have no influence on the convergence in the case of the Reber-like target and therefore they are not in the focus of this section. Nevertheless, we added them in order to make our dataset hybrid (i.e. to include both discrete and continuous variables). They are randomly generated according to a Gaussian distribution. Output and input signals have a mean value and a standard deviation of respectively 12.6 ± 1.26 and 2.2 ± 0.22.

The HyBUTLA algorithm receives the generated data as an input. The results are given graphically in Figure 9.2. They are obtained for a value of the confidence parameter $\alpha = 0.01$ (this parameter is needed for computation of the function *fractions-different* given by the expression (6.3) in Section 6.3). The plot shows the number of states of the hypothesis automaton as a function of the number of learning examples. It can be seen that when the size of the input data is large enough, the number of states always converges to the correct value defined by the Reber-like

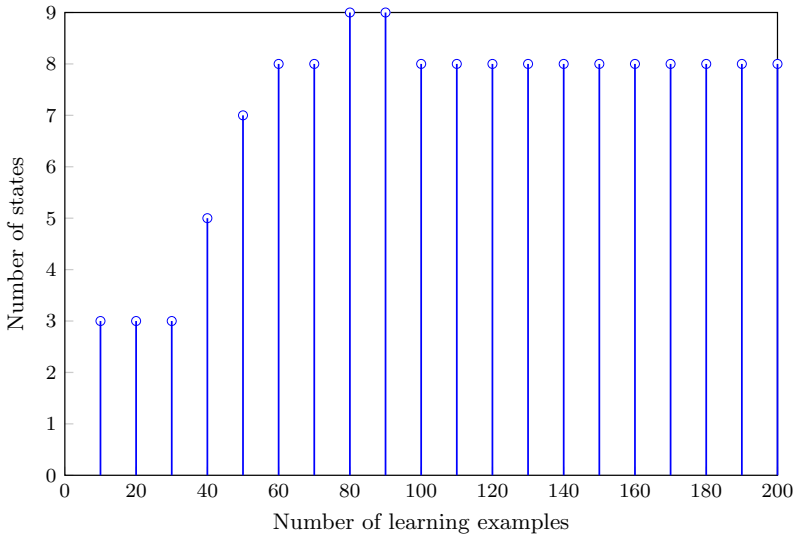


Fig. 9.2 Convergence of the HyBUTLA to the target Reber-like automaton for $\alpha = 0.01$.

automaton (i.e. to eight states, see Figure 9.1). Furthermore, the convergence is achieved with a relatively small number of learning examples (about one hundred). Moreover, the transition and ending state probabilities are correctly inferred provided any number of examples greater than or equal to one hundred.

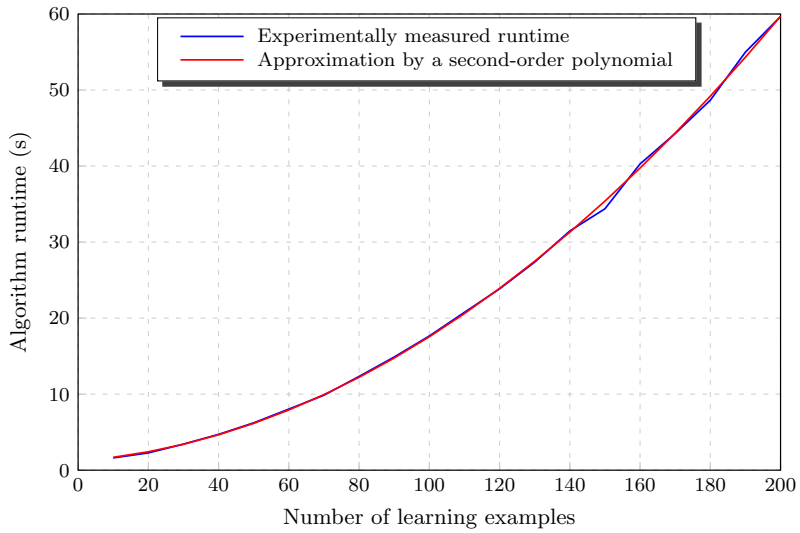


Fig. 9.3 Runtime of the HyBUTLA algorithm for $\alpha = 0.01$ and its quadratic approximation.

In addition to convergence, we have also measured the algorithm runtime. Our idea was to validate the HyBUTLA theoretical polynomial runtime result experimentally. As given by Lemma 7 in Section 6.6, its time complexity is $O(n^3)$, where n is the size

of the input data. The empirical runtime, measured on a PC with the Intel(R) Xeon(R) W3503 CPU @ 2.40 GHz and 6 GB RAM that runs a MATLAB implementation of the HyBUTLA, is shown in Figure 9.3 (again for $\alpha = 0.01$). We made an approximation of the runtime curve by a second-order polynomial. It can be seen that even for larger sizes of the input data, the runtime is quadratic in the average case. This confirms our theoretical result.

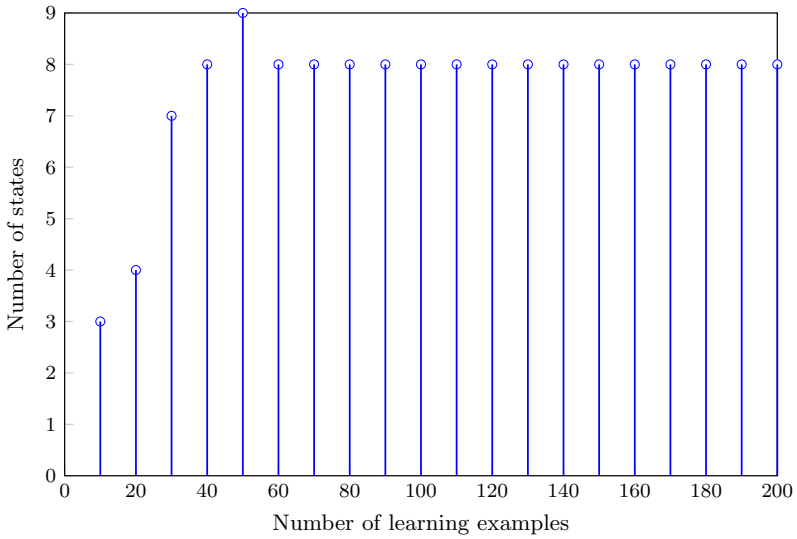


Fig. 9.4 Convergence of the HyBUTLA to the target Reber-like automaton for $\alpha = 0.1$.

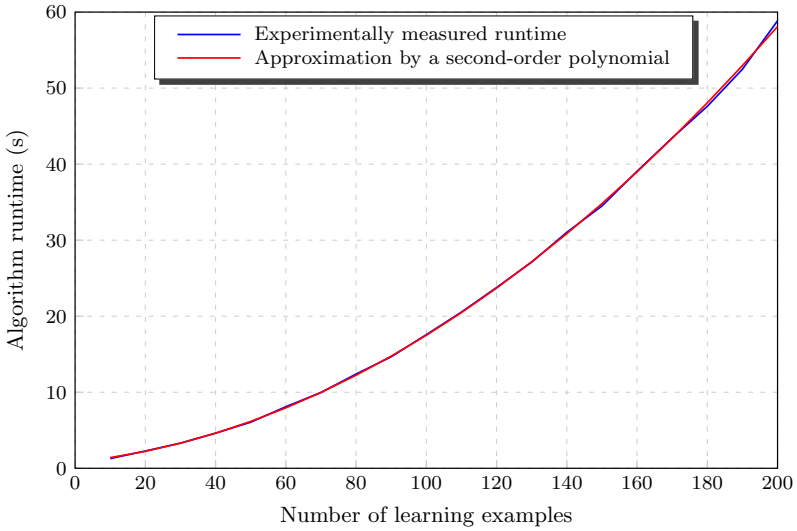


Fig. 9.5 Runtime of the HyBUTLA algorithm for $\alpha = 0.1$ and its quadratic approximation.

In Section 6.6, we made the observation that by increasing the value of the α parameter, the convergence speed can be increased as well (Observation 8). In order to experimentally show this trend, we repeated the convergence experiments for α = 0.1. The convergence to the Reber-like automaton and the empirical runtime of the algorithm are given in Figure 9.4 and Figure 9.5, respectively.

By increasing the value of α by factor ten, the convergence was reached significantly faster, i.e. with only 60 learning examples. Already at that point, both the automaton structure and all of its probabilities were correctly identified. The runtime of the algorithm is still quadratic in the size of the input data, being slightly shorter than in the previous case.

9.2 Empirical Analysis of Scalability

The goal of the scalability experiments is to evaluate the HyBUTLA algorithm performance in the presence of an increasing number of two types of signals in the system: discrete and continuous. A number of experiments were conducted by increasing the number of one type of signals, while keeping the number of the other type constant. We first give the overall setting of the experiments in the following subsection. Then we present the results obtained on datasets with a constant number of continuous signals (Subsection 9.2.2). They are followed by the results for a constant number of discrete signals (Subsection 9.2.3). The results are given graphically for the *Prefix Tree Acceptor* (PTA), *merged hybrid automaton* (MERGE step), and *split hybrid automaton* (SPLIT step). The given performance metrics include the number of PTA states, the number of merges, the number of splits, the average model coefficient of determination (R^2), the size reduction (in relation to PTA size), and the learning time. We also give some general results that bound the maximum number of merges and the minimum number of splits that the HyBUTLA algorithm can perform under given conditions.

9.2.1 The Setting of the Experiments

The setting of the experiments is explained here briefly. In total, 88 artificial datasets were generated. Each dataset comprises 10 learning examples. The size of each learning example was picked from a range of $[150, 250]$ samples, with a random number generator that uses an uniform distribution. Multiple linear regression with linear terms was used as a regression method of the HyBUTLA algorithm. The experiments were conducted using the same PC as in Section 9.1 (CPU @ 2.40 GHz and 6 GB RAM).

A normal distribution was used for generating both the continuous input signals and the output signal. The mean value and standard deviation for input signals is 220 ± 22 and for the output signal 1206 ± 120.6.

Discrete signals were generated following an uniform distribution. They represent independent binary variables. Locations and lengths of bit-switches are picked randomly for every signal. Each discrete signal changes two times in every learning example.

In all experiments, we kept the number of the continuous output signals constant at the value one, while we varied the number of both discrete and continuous input signals. For easier reading, the number of continuous input signals will be denoted by c, and the number of discrete signals by d.

9.2.2 Analysis with a Constant Number of Continuous Signals

For each of the three automaton structures (PTA, merged and split automaton), four sets of experiments were conducted. The number of discrete signals (d) has been increased up to 50, while keeping the number of continuous signals (c) constant at values: 1, 5, 25, and 50. We give the results graphically by the corresponding plots.

9.2.2.1 Prefix Tree Acceptor (PTA) Results - Constant c

Figure 9.6 gives the PTA results for four different numbers of continuous signals in the system (1, 5, 25, and 50).

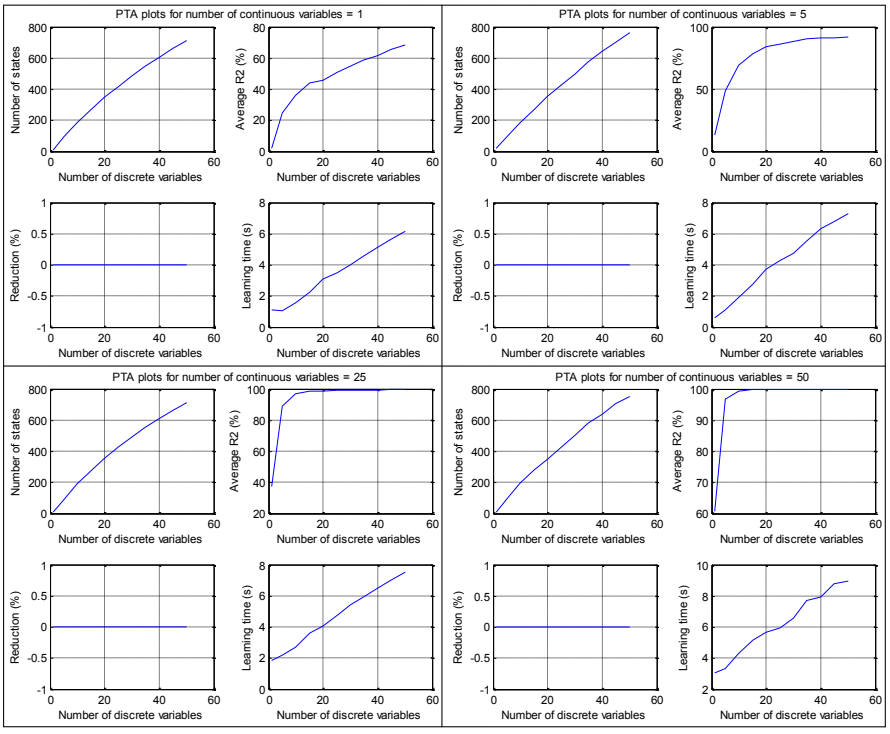


Fig. 9.6 Constant number of continuous signals - PTA results.

It can be seen that the number of PTA states *#PTAstates* grows at most linearly with the number of independent discrete signals. Since the number of learning examples is kept constant across different experiments and their size is always taken from the

same range ($[150, 250]$), the increase in d generates more states and thus the number of samples per each state decreases. This makes it easier to approximate the output signal by using the inputs, thus the average R^2 rises with d for all four values of c. The size reduction of the final model with the number of states denoted by *#states* is computed in relation to the PTA size *#PTAstates*, i.e. by using the expression (*#PTAstates* - *#states*) $\cdot 100$ / *#PTAstates*. Since in these experiments we only created prefix trees and no merging took place, *#PTAstates* and *#states* were always equal. Therefore, the size reduction was always equal to zero in PTA experiments. This can be seen in the reduction diagrams for all four values of c. In addition, in all four cases we observe that the learning time increases approximately linearly with the increase of d.

9.2.2.2 Merged Hybrid Automaton (MERGE step) Results - Constant c

Results for the merged hybrid automaton, obtained after applying the MERGE step on PTA models, are summarized in Figure 9.7. For all four values of c, it can be observed that the number of merges increases with d. In Subsection 9.2.4 we will bound this growth with a linear function.

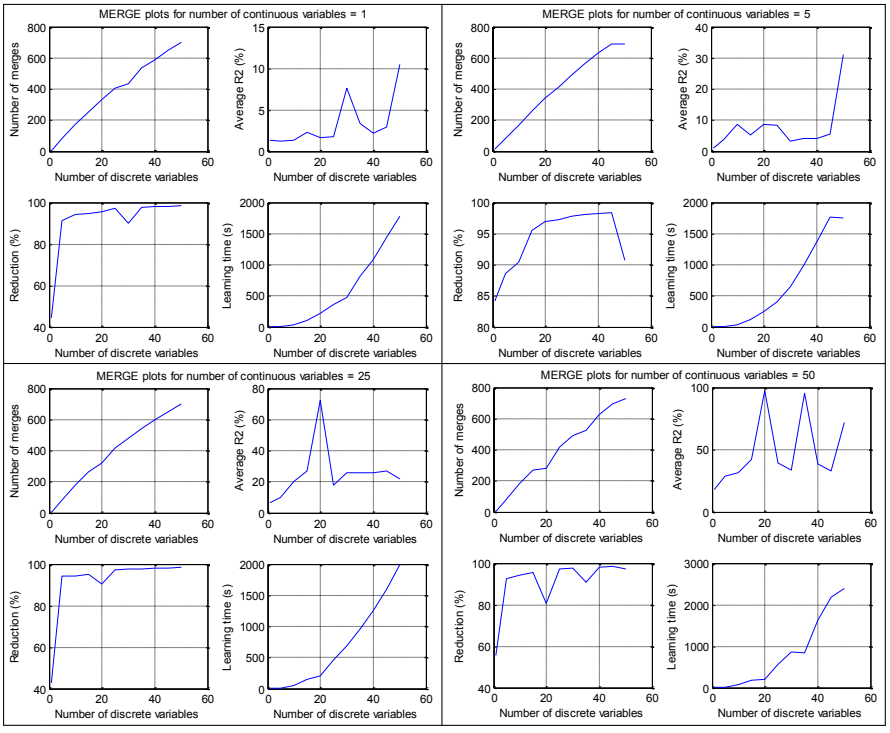


Fig. 9.7 Constant number of continuous signals - MERGE results.

As more states are merged, their corresponding continuous datasets are combined, making it more difficult to approximate the output signal. Therefore, the MERGE step decreases the average R^2 in general (see Figure 8.3 in Section 8.2). The increase of

d has the opposite effect (as stated in the previous subsection). Therefore, significant variations of the average R^2 values are recorded in these experiments, based on what effect was dominant for a particular model (the merging or the increase of d). For example, when $c = 5$, the number of merges gets constant for d greater than 45. Thus, the increase of d is for matching models more dominant than the number of merges and recorded average R^2 values are correspondingly much larger for these models. Similarly, peaks in average R^2 values can be observed in diagrams for $c = 1$, $c = 25$ and $c = 50$ exactly for values of d that correspond to somewhat lower number of merges. That is, wherever fewer merges are observed, the average R^2 obtains higher values.

In Figure 9.6 we saw that the number of PTA states increases approximately linearly with d. When merging states, the size reduction is computed in relation to the PTA size, as shown in the previous subsection. Therefore, whenever fewer merges are performed for some value of d, a drop is observed in the size reduction for that value. This trend can nicely be seen in Figure 9.7 for $c = 5$ when the number of merges is constant. Since the number of PTA states for the corresponding values of d is growing and no merges are occurring, the drop can be seen in size reduction.

The experimental learning time is for all values of c subquadratic.

9.2.2.3 Split Hybrid Automaton (SPLIT step) Results - Constant *c*

The SPLIT step is applied on merged hybrid automata. The graphical results for this step are given in Figure 9.8.

The number of splits depends on the fulfillment of the conditions described in Subsection 6.5.2. However, it can be noted that the number of splits is high for lower values of d and it decreases as d increases. Eventually, it drops to zero for large d values. We have analyzed this trend in Section 9.2.4.

Since the *split* function is applied in order to increase the average R^2, this measure is in direct connection to the number of splits. For models with higher number of preformed splits, significantly higher average R^2 values are observed. This can be seen in diagrams for all four values of c in Figure 9.8. For example, for $c = 25$ the peak in the number of splits exists for $d = 20$. Correspondingly, the peak is observed in the average R^2 values for $d = 20$.

The size reduction is inversely proportional to the number of splits. More splits produce more states, thus lower the model size reduction in relation to the PTA size. Sometimes a negative reduction is obtained, which means that the split hybrid automaton is bigger in the number of states than the initial PTA model. In the example for $c = 25$ and $d = 20$, a drop is recorded in the size reduction, since there is a peak in the number of splits for that particular model.

The learning time is of course longer where more splits are performed and in general, it grows with d. In the aforementioned example with the peak in the number of splits ($c = 25$, $d = 20$), longer learning time is needed for obtaining that particular model.

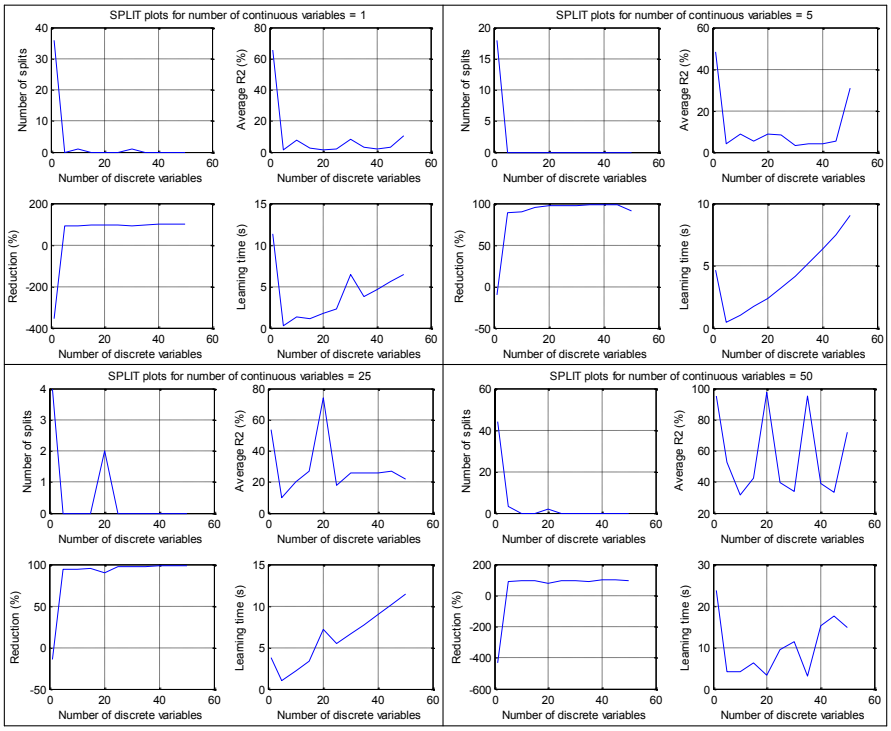


Fig. 9.8 Constant number of continuous signals - SPLIT results.

9.2.3 Analysis with a Constant Number of Discrete Signals

In this analysis, the number of continuous input signals (c) has been increased up to 50, while keeping the number of discrete signals (d) constant. Similarly to the previous analysis, four sets of experiments were conducted for each of the three automaton structures (PTA, merged and split automaton), namely for keeping d constant at values: 1, 5, 25, and 50. The results are given graphically.

9.2.3.1 Prefix tree acceptor (PTA) results - constant d

Results for the PTA are given in Figure 9.9. Since new states of the PTA are generated solely based on changes in discrete signals, the increase of c does not affect the number of PTA states. This number remains constant for all four values of d shown in the figure.

However, c affects the average R^2 of the obtained models, since more predictors can approximate the output variable more easily. For all four cases (when d equals 1, 5, 25, and 50), it can be seen that the average R^2 grows with the number of predictors c.

Since in the PTA experiments (just as shown in Subsection 9.2.2.1) the number of PTA states equals the number of the states in the final model (i.e. PTA is the final model), the size reduction maintains the zero value.

The more predictors are present in the system, the more time is needed for learning the model θ functions. For all four values of d, the runtime of the algorithm grows with c. We emphasize that the time was measured experimentally on a PC running multiple processes. That accounts for certain small variations in the recorded runtime of the algorithm. Nevertheless, the experiments clearly indicate the existing relation between the algorithm runtime and the number of continuous signals c.

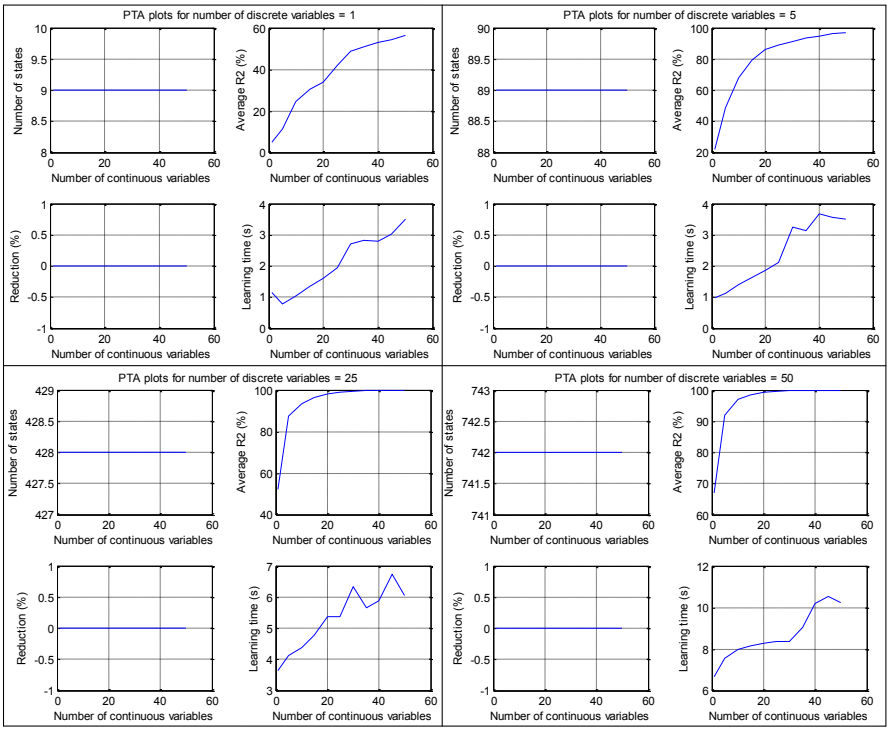


Fig. 9.9 Constant number of discrete signals - PTA results.

9.2.3.2 Merged Hybrid Automaton (MERGE step) Results - Constant d

The results of the MERGE step for constant d are shown in Figure 9.10.

The merging criteria include checking the similarity in probabilities of transitions and ending states, and it is not influenced by the continuous signals in the system. Therefore, the number of merges remains constant with the growth of c in all four sets of experiments (i.e. for d kept constant at values: 1, 5, 25, and 50). Consequently, the size reduction remains constant as well.

Here the average R^2 also increases with the growth of c, as it is easier to approximate the output signal with more predictors included in the merged automaton model, just as it was the case with the PTA. We can also observe that the relation between the average R^2 and c is approximately linear.

The similarity to PTA results exists also in the learning time, i.e. more time is needed for learning models with larger c.

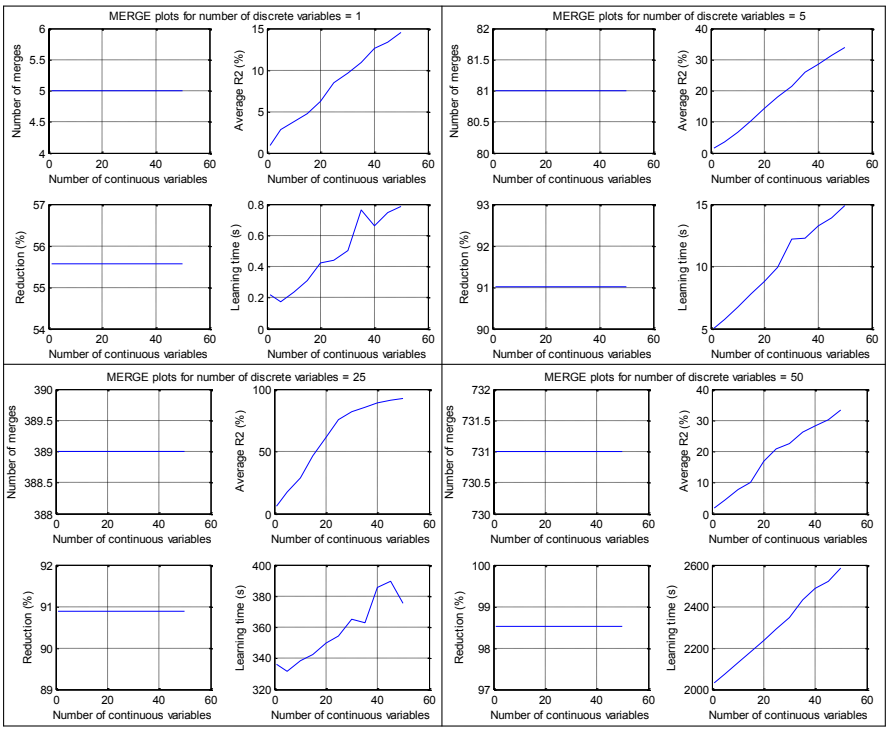


Fig. 9.10 Constant number of discrete signals - MERGE results.

9.2.3.3 Split Hybrid Automaton (SPLIT) Results - Constant d

Figure 9.11 presents the results of the SPLIT step for the constant number of discrete signals d.

Interestingly, the significant number of splits is obtained only in two sets of experiments, namely for $d = 1$ and $d = 5$. For $d = 25$ only one split is performed and for $d = 50$, there were no splits. The same trend is observed in Figure 9.8. When the number of discrete signals in the system is too large, fewer or no splits can be performed. This trend is analyzed in the following section. The number of splits also depends on the fulfillment of the *split* function criteria, described in Subsection 6.5.2.

As in the previous two cases (PTA and MERGE step), average R^2 of the models obtained with the SPLIT step also grows with c. Moreover, average R^2 gets additionally increased when more splits are performed.

The size reduction is inversely proportional to the number of splits. This can be seen in diagrams for $d = 1$ and $d = 5$. The more splits are performed, the more new states are generated in the final model. This lowers the size reduction. For $d = 1$ it can be seen that the size reduction gets negative, which means that the number of states in final models created by the *split* function got bigger than the number of states in the initial prefix trees. The size reduction remains constant for constant number of splits, which can be seen in diagrams for $d = 25$ and $d = 50$.

Again, the learning time generally grows with c. We emphasize again that the experimentally measured learning time was influenced by other processes running on the same PC.

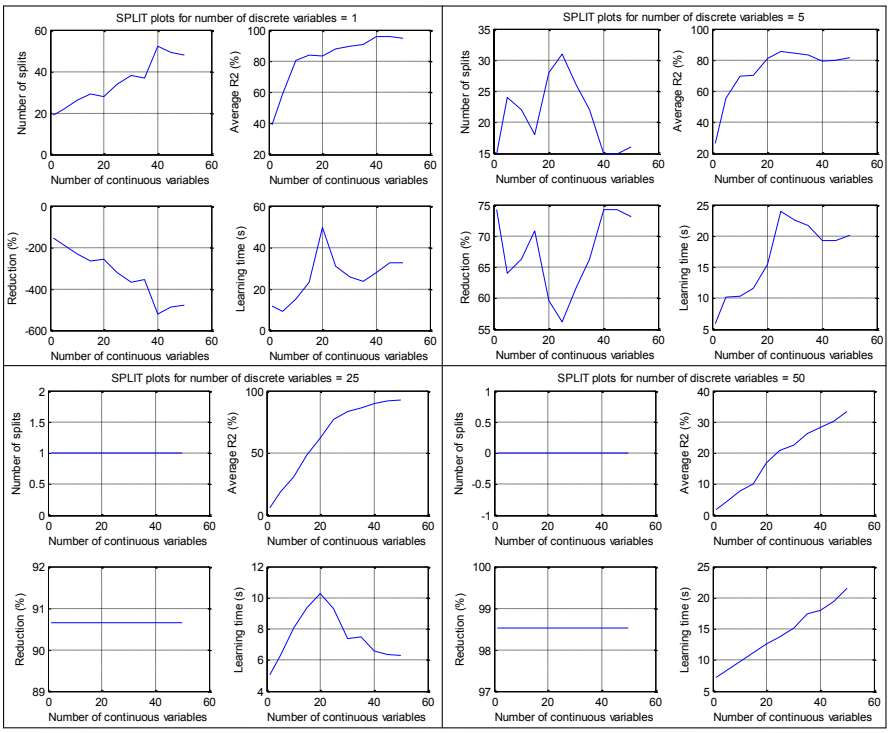


Fig. 9.11 Constant number of discrete signals - SPLIT results.

9.2.4 General Scalability Results

Here we show some general dependencies between the numbers of signals in a system and the model-learning performance. First, in Theorem 16 we give the maximum number of merges that can be performed under the given conditions. Then in Theorem 17 we show that if the number of discrete signals in the system is equal to or greater than the given bound, no splits of the automaton states can be performed.

Theorem 16. *Assume a set $\mathcal{D} = \{D_1, D_2, ..., D_k\}$[1] of k learning examples is given, where each example contains d mutually independent discrete signals (change in one signal does not affect changes in other signals). Then for the maximum number of merges M_{max} of the automaton states it holds:*

$$M_{max} \leq kz_{max}d,$$

where z_{max} denotes the maximum number of changes (bit-switches) over all discrete signals in all learning examples.

Proof. Since the PTA transitions are triggered by the changes in discrete signals only, let us assume the worst case where every change in every discrete signal and in every example triggers a transition to a new state, which was not triggered before. This is the worst case, because the number of PTA states is then maximized and that

[1] The learning example is illustrated in Section 3.4, Example 1.

is contrary to the general requirement of having models as small as possible. Further let z_j^i denote the number of changes of the jth discrete signal in the ith learning example, where $i = 1, ..., k$, and $j = 1, ..., d$. The number of PTA states $t(D_i)$ that is generated by the example D_i is then:

$$t(D_i) = \sum_{j=1}^{d} z_j^i. \tag{9.1}$$

This case is illustrated in Figure 9.12. It shows the example D_i with the number of timed samples $l = 7$. The first column is the time, next three columns are discrete binary signals q_1, q_2 and q_3 (i.e. $d = 3$), last two columns are continuous input and continuous output signals, respectively. The signal q_3 changes once (blue frame in Figure 9.12) and thus generates the state s_1. The signal q_2 changes twice (green frames), generating states s_2 and s_3. The states s_4, s_5 and s_6 are generated by three changes in signal q_1 (red frames). The state s_0 is the initial state, which is shared between all examples D_i, $i = 1, ..., k$. For the values of signals q_1, q_2 and q_3 in D_i, it holds respectively: $z_1^i = 1$, $z_2^i = 2$ and $z_3^i = 3$.

$$D_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 9.6 & 14.5 \\ 2 & 0 & 0 & 1 & 10.5 & 15.2 \\ 3 & 0 & 1 & 1 & 12.4 & 14.9 \\ 4 & 0 & 0 & 1 & 7.5 & 10.6 \\ 5 & 1 & 0 & 1 & 6.6 & 9.2 \\ 6 & 0 & 0 & 1 & 6.5 & 9.2 \\ 7 & 1 & 0 & 1 & 6.4 & 9.1 \end{bmatrix} \begin{matrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{matrix}$$

Fig. 9.12 States generated by the example D_i.

The total number of PTA states t_n, triggered by all k examples is as follows:

$$t_n = t(\mathcal{D}) = \sum_{i=1}^{k} t(D_i) + 1 = \sum_{i=1}^{k} \sum_{j=1}^{d} z_j^i + 1. \tag{9.2}$$

Additive factor '1' accounts for the initial state. The maximum number of merges M_{max} is obtained when the merging criteria hold for every pair of states, i.e. when the whole PTA is merged into a single state. In that case, we have:

$$M_{max} = t_n - 1. \tag{9.3}$$

Combining (9.2) and (9.3), it follows:

$$M_{max} = \sum_{i=1}^{k} \sum_{j=1}^{d} z_j^i. \tag{9.4}$$

As stated in the theorem, z_{max} denotes the maximum number of changes (bit-switches) over all examples and signals, i.e.

$$z_{max} = \max_{i,j}\{z_j^i\}. \tag{9.5}$$

From (9.4) we finally obtain:

$$M_{max} \leq \sum_{i=1}^{k}\sum_{j=1}^{d} z_{max} = k z_{max} d, \tag{9.6}$$

thus the theorem is proven. □

Example 5. *Using the expression given by (9.6), we can calculate the theoretical maximum number of merges for all four datasets given in Figure 9.10. For values of* d *that equal to* 1*,* 5*,* 25*, and* 50*, we get* M_{max} *values of* 20*,* 100*,* 500*, and* 1000 *respectively (in our experiment setting* $z_{max} = 2, k = 10$*). Figure 9.10 shows experimentally obtained number of merges, respectively:* 5*,* 81*,* 389*, and* 731*. Theoretical maxima are never violated.*

Theorem 17. *Assume a set* $\mathcal{D} = \{D_1, D_2, ..., D_k\}$ *of* k *learning examples is given, where each example contains* d *mutually independent discrete signals (change in one signal does not affect changes in other signals). Then the number of splits* N *of the automaton states equals to zero for sufficiently large* d*.*

Proof. The goodness of fit of approximating the continuous output signal in an automaton state is expressed as the coefficient of determination $R^2(\theta)$, which shows how good the learned θ function represents the observed data y_j. It is given by the expression (6.1) in Section 6.3 but for easier reading we repeat it here:

$$R^2(\theta) = 1 - \frac{SS_{error}}{SS_{total}} = 1 - \frac{\sum\limits_j (y_j - \theta_j)^2}{\sum\limits_j (y_j - \overline{y})^2},$$

where SS_{error} is the sum of squares of errors, SS_{total} is the total sum of squares, θ_j are values of learned θ function at j points, and $\overline{y}$ is the mean value of observed data y_j. The average coefficient of determination of the whole model is calculated according to:

$$R_A^2 = \frac{\sum\limits_{h=0}^{w-1} R^2(\theta_{s_h})}{w},$$

where w denotes the number of automaton states. The higher R_A^2, the fewer splits are needed and vice versa. Therefore, the connection between these variables has the following form:

$$N = C(1 - R_A^2), \tag{9.7}$$

where $C \in (0, \infty)$ is an unknown multiplicative factor. Let LS denote the size (number of samples) of the longest learning example, i.e.

$$LS = \max_i\{l_i\}, \tag{9.8}$$

where l_i is the length of i-th example. Since all k examples share the initial state, for the number of automaton states t_n it holds:

$$t_n \leq k(LS - 1) + 1. \tag{9.9}$$

Similarly to the previous proof, let us assume the worst case scenario where split can be performed after every sample of every learning example. In this way, all created states will have only one corresponding sample. We call such states *dummy* states. This setting is illustrated in Figure 9.12 for one example D_i. Looking at the expression for calculating $R^2(\theta)$, it is clear that for dummy states there is only 1 data point for approximation and thus it holds:

$$y_j = \theta_j \Rightarrow R^2(\theta) = 1. \tag{9.10}$$

From (9.3), (9.6) and (9.9) it follows that the maximum number of created (dummy) states is:

$$t_n = kz_{max}d + 1 = k(LS - 1) + 1, \tag{9.11}$$

where z_{max} is maximum number of changes (bit-switches) over all signals and examples. During the SPLIT step, R_A^2 approaches to 1 as the number of states w approaches t_n, so it follows:

$$\lim_{w \to kz_{max}d+1} R_A^2 = \lim_{w \to kz_{max}d+1} \frac{\sum_{h=0}^{w-1} R^2(\theta_{s_h})}{w} = \frac{\sum_{h=0}^{kz_{max}d} R^2(\theta_{s_h})}{kz_{max}d + 1} = 1, \tag{9.12}$$

as every $R^2(\theta_{s_h}) = 1$ for $w = kz_{max}d + 1$ (see (9.10)). In this case, the number of splits given by (9.7) equals to zero. From the expression (9.11) follows the number of discrete signals for which $N = 0$:

$$d \geq \frac{t_n - 1}{kz_{max}} = \frac{k(LS - 1) + 1 - 1}{kz_{max}} = \frac{LS - 1}{z_{max}}. \tag{9.13}$$

This completes the proof. □

Observation 11. *In practical cases we do not aim at unrealistic $R_A^2 = 1$, but rather at R_A^2 greater or equal to some given threshold. Therefore, we do not allow creation of dummy states by the SPLIT step.*

Example 6. *Using the expression (9.13), we can calculate the minimum number of discrete signals for which the number of splits gets the zero value. Since in the experiments we did not allow creation of dummy states, but allowed minimum of 3 samples per automaton state, the maximum number of obtained states is $t_n = \lfloor \frac{k(LS-1)+1}{3} \rfloor$. The expression (9.13) now becomes:*

$$d \geq \frac{t_n - 1}{kz_{max}} = \frac{\lfloor \frac{k(LS-1)+1}{3} \rfloor - 1}{kz_{max}}.$$

For our experimental setting ($k = 10, LS = 250, z_{max} = 2$), we get:

$$d \geq \frac{\lfloor \frac{10 \cdot (250-1)+1}{3} \rfloor - 1}{10 \cdot 2} \approx 42.$$

Looking at Figure 9.8, but also at Figure 9.11, we can verify that the number of splits N is always zero for $d \geq 42$.

9.3 Conclusion

In this chapter, we have used artificial datasets to demonstrate the convergence, polynomial runtime and scalability properties of the HyBUTLA algorithm.

The convergence of the algorithm has been shown on the typical benchmark target automaton with only eight states (the Reber-like model). The algorithm learned the correct automaton using the relatively small number of learning examples (60–100 examples, based on the value of α parameter of the Hoeffding bound). Moreover, the experimentally measured runtime of the algorithm is quadratic in the size of the input data.

The HyBUTLA scalability was also evaluated using artificially generated datasets. Experiments showed the influence that the numbers of continuous (c) and discrete (d) signals in the system have on the model-learning performance. The first general result (Theorem 16) is that the maximum number of merges of PTA states is bounded by a function linear in d, assuming that discrete signals are independent (the change in one signal does not affect other signals). The second result (Theorem 17) shows that the number of state splits goes to zero for sufficiently large d. Both these findings are formally proven. Based on both experimental and theoretical results, several general observations important for modeling hybrid production systems with the HyBUTLA algorithm in practice can be derived. These are illustrated in Table 9.1 and explained in the following.

Table 9.1 General observations for modeling hybrid production systems with the HyBUTLA algorithm.

Dominantly discrete system (larger d, smaller c)	Dominantly continuous system (smaller d, larger c)
larger PTA	smaller PTA
many merges	fewer or no merges
higher size reduction	lower size reduction
fewer or no splits	many splits
lower average R^2	higher average R^2

For dominantly discrete systems, one can expect to obtain large PTAs, but at the same time to benefit a lot from the MERGE step in the sense of size reduction. Very small models (high reduction rates) could be obtained. Unfortunately, this typically produces lower accuracy of approximating continuous output signals (low average R^2). In this case, the SPLIT step cannot significantly increase average R^2 due to the large d values, which cause fewer or no splits.

For dominantly continuous systems, the situation is converse. With smaller d, PTAs of the small size are obtained. Larger c does not influence the PTA size and the number of merges. A small d enables very few or no merges, thus the MERGE step does not bring significant benefit in modeling such systems, in the sense of size reduction. However, a significant number of SPLITS can be expected and thus higher average R^2 values. The trade-off in this case is the increased model size, as the SPLIT step can produce models several times larger than the initial PTA.

Part V

Conclusion

Chapter 10

Conclusions and Future Work

In this chapter, we shortly present the main conclusions and contributions of this thesis. Every conducted research answers certain opened questions, but also opens the new ones. Our research presented in this thesis is no exception, and therefore we give an outlook of future work.

10.1 Conclusions

This thesis was started with the example of the dangerous abnormal behavior of the production process in the local chemical plant in Niihama city in Japan. This and a number of other incidents in industrial facilities emphasized the need of having reliable anomaly detection and diagnosis approaches. Most of such approaches today are based on manually created behavior models. Since manual modeling is a very hard task, this thesis offers an alternative that comes from the field of computational learning theory.

We investigated the identification of behavior models for production systems automatically from data. The nature of such systems is hybrid, as they exhibit state-based (discrete), continuous, timed, and probabilistic behavioral aspects. Due to the well-founded theory for various finite automata and their ability to represent characteristics of different technical systems, we focused our attention on learning automata behavior models. Our ultimate goal was to develop an approach for learning *hybrid automata*, which can adequately model hybrid production systems.

First, we gave an overview of systems and models, focusing on several types of finite automata in Chapter 2. Additionally, we cited a number of existing anomaly detection and diagnosis approaches that are based on hybrid automata (and similar) models. In contrast to the number of these approaches, there is to date no algorithm that can learn hybrid automata automatically from data.

By generalizing the existing results of learning a few classes of deterministic automata, we have showed in Chapter 3 that the deterministic hybrid automata with one clock that tracks the continuous time evolution and resets at every transition can be learned from positive and negative examples (i.e. logs taken during normal and abnormal plant operation) in the limit (i.e. when enough data are available).

However, due to the inability of such automata to model stochastic behavior of a plant, but also due to the lack of negative learning examples in practice, we have investigated the identifiability of stochastic automata only from positive examples in Chapter 4. Several algorithms exist that can learn stochastic deterministic finite automata and one-clock stochastic deterministic timed automata from positive examples in the limit with probability one (i.e. learning the correct target automaton). Moreover, their runtime is polynomial in the size of the input data. The key property that makes the stochastic automata identifiable in the limit with probability one from positive examples is that they come from well-defined probability distributions [Ang88a]. This property compensates for the lack of negative examples. These existing positive results directed our research to identifying the *One-clock Stochastic Deterministic Hybrid Automata* (1-SDHAs).

In Chapter 6, we presented our *Hybrid Bottom-Up Timing Learning Algorithm* (HyBUTLA) for learning 1-SDHAs automatically from positive data in the limit with probability one. We proved its convergence properties and showed that its runtime is polynomial in the size of the input data. The negative result that even the simplest stochastic deterministic finite automata cannot be learned from data of the size polynomial in the size of the automaton, unfortunately generalizes to 1-SDHAs. A potential workaround for this issue is the framework of *Probably Approximately Correct* learning (PAC-learning) which enables identification of approximately correct automata from polynomial data (see discussion in Chapter 5).

In addition to the learning algorithm, we have presented our *ANOmaly Detection Algorithm* (ANODA) in Chapter 7. It uses the automatically learned behavior models for model-based anomaly detection. We have proven its real-time properties that enable its online application (i.e. during the runtime of the monitored system).

The real-world applicability of our model-learning and anomaly detection approaches has been demonstrated in two real plants in Chapter 8. Promising experimental results were obtained.

Experimental validation of the HyBUTLA convergence, polynomial runtime and scalability properties has been performed using the artificially generated datasets in Chapter 9.

In the following, we summarize the main contributions of this thesis:

- We have found a formalism of 1-SDHAs to be both learnable from data and suitable for modeling main characteristics of hybrid production systems, such as interacting discrete-continuous dynamics, timed and stochastic behavior.
- We have developed the first hybrid automata learning algorithm: the HyBUTLA algorithm.
- We positioned our learning approach in the larger picture of the existing relevant complexity results. We showed that the HyBUTLA algorithm learns 1-SDHAs from positive examples in the limit with probability one in polynomial time.
- The expert knowledge required for our approach is identified. The HyBUTLA algorithm uses typically available knowledge, comprising the information about: (i) the structure of the system, (ii) the associated signals for every modeled component, and (iii) logs of those signals. The ANODA algorithm requires the following knowledge: (i) the system sampling rate, (ii) the algorithm response time for considered system, (iii) measurement ranges for monitored continuous output variables, and (iv) a predefined threshold for signaling the alarm.
- We conducted the extensive experiments in two real plants as well as on the artificial data. They empirically confirm the HyBUTLA convergence, polynomial

runtime and scalability properties, but also show the applicability of our approach in the real-world.

10.2 Future Work

During this research, we have identified several possible directions of future work. We explain them in the following:

Algorithm optimization: In its current state, the HyBUTLA algorithm learns functions that approximate the continuous output signals (θ functions) of the system in several steps. First, the functions are learned for all prefix tree states. Then, whenever two automaton states are merged, functions are learned for the newly created state. Finally, whenever a state is split, functions need to be learned for the two obtained states. In this way, functions are often learned for those states that will be merged or split in subsequent algorithm steps. It would be possible to speed up the algorithm by only maintaining the pointers to data required for learning the θ functions of automaton states, and to learn all those functions only at the end, after both merge and split steps are completed.

PAC-learning of hybrid automata: The number of positive examples needed for learning 1-SDHAs in the limit with probability one cannot be in general bounded by a polynomial in the size of the target automaton. However, we are confident that learning of 1-SDHAs from a polynomial amount of data is possible in the PAC-learning framework. The price to pay is the fact that in this framework, instead of learning the correct automaton, a probably approximately correct automaton is learned, i.e. a small error between the target and the learned automaton will exist with certain probability.

Modeling non-production hybrid systems: In this work, we focused exclusively on learning models for hybrid production systems, such as process plants. In such facilities, more empirical analyses can be performed using our approach, closing all the gaps that our own analyses might have left open. However, there are also other types of hybrid systems that suffer from known drawbacks of manual modeling. An example is the electric car, which also comprises discrete, continuous, timed, and stochastic behavioral aspects.

Different application areas: We have evaluated our approach in the application of model-based anomaly detection. In Section 1.4 we have listed a number of other possible application areas, such as the model-based design, testing and optimization. We think that models, automatically learned by our HyBUTLA algorithm, could in many cases replace manually created behavior models used in these applications.

Extending the ANODA algorithm: The ANODA algorithm uses the behavior model learned by the HyBUTLA algorithm in order to detect the anomalous behavior, i.e. the faults in discrete control and continuous physical system. Model-based diagnosis however includes both fault detection and fault isolation. This means that a reasoning mechanism exists, which isolates a faulty component and finds a concrete failure that caused the anomalous behavior. By adding such a mechanism to the ANODA algorithm, it could be updated from model-based anomaly detection to model-based diagnosis algorithm.

Abbreviations

1-DTA	One-clock Deterministic Hybrid Automaton	38
1-DTA	One-clock Deterministic Timed Automaton	38
1-SDHA	One-clock Stochastic Deterministic Hybrid Automaton	4
1-SDTA	One-clock Stochastic Deterministic Timed Automaton	54
ANODA	ANOmaly Detection Algorithm	4
BUTLA	Bottom-Up Timing Learning Algorithm	29
CLHA	Cycle-Linear Hybrid Automaton	29
CPHA	Concurrent Probabilistic Hybrid Automaton	31
CWT	Continuous Wavelet Transform	87
DES	Discrete Event System	12
DFA	Deterministic Finite Automaton	17
DHA	Deterministic Hybrid Automaton	18
DTA	Deterministic Timed Automaton	17
DWT	Discrete Wavelet Transform	85
EDESA	Extended Discrete Event System Abstraction	33
EM	Expectation-Maximization	32
FDD	Fault Detection and Diagnosis	1
FN	False Negative	122
FP	False Positive	121
FT	Fourier Transform	86
HBG	Hybrid Bond Graph	33
HMM	Hidden Markov Model	31
HyBUTLA	Hybrid Bottom-Up Timing Learning Algorithm	4
K-L divergence	Kullback-Leibler divergence	27
LMF	Lemgo Model Factory	4
LOOCV	Leave-One-Out Cross Validation	125
MBD	Model-Based Diagnosis	1
MDI	Minimal Divergence Inference	27
MLR-LT	Multiple Linear Regression with Linear Terms	117
NFA	Non-deterministic Finite Automaton	17
NHA	Non-deterministic Hybrid Automaton	46
NTA	Non-deterministic Timed Automaton	46
OPC-UA	OPC Unified Architecture	75
PAC-learning	Probably Approximately Correct learning	21
PDRTA	Probabilistic Deterministic Real-Time Automaton	19
PLC	Programmable Logic Controller	14
PTA	Prefix Tree Acceptor	24
RLIPS	Regular Language Inference from Positive Samples	27
RPNI	Regular Positive and Negative Inference	23

RTI	Real-Time Identification	28
RTI+	Real-Time Identification from Positive Data	28
RTS	Real-Time System	108
SDFA	Stochastic Deterministic Finite Automaton	18
SDHA	Stochastic Deterministic Hybrid Automaton	19
SDTA	Stochastic Deterministic Timed Automaton	19
TCG	Temporal Causal Graph	33
TN	True Negative	122
TP	True Positive	121
WT	Wavelet Transform	86

List of Figures

List of Tables

References

[AAB09] V. Alarcon-Aquino and J. A. Barria. Change detection in time series using the maximal overlap discrete wavelet transform. 39(2):145–152, 2009.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. h. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AL88] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, April 1988.

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987.

[Ang88a] D. Angluin. Identifying languages from stochastic examples. Technical report, Yale University, YALEU/DCS/RR-614, March 1988.

[Ang88b] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[APR02] J. Abello, P. M. Pardalos, and M. G. C. Resende, editors. *Handbook of massive data sets*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[Aut10] Automation Markup Language AutomationML. www.automationml.org, 2010.

[BHR08] P. Bouyer, S. Haddad, and P. A. Reynier. Timed petri nets and timed automata: On the discriminating power of zeno sequences. *Information and Computation*, 206(1):73 – 107, 2008.

[BN93] M. Basseville and I. V. Nikiforov. *Detection of abrupt changes: theory and application*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[Boy08] S. A. Boyer. *SCADA: Supervisory Control And Data Acquisition*. Instrument Society of America, North Carolina, USA, 2 edition, 2008.

[Bra05] M. S. Branicky. Introduction to hybrid systems. In *Handbook of Networked and Embedded Control Systems*, pages 91–116. Birkhauser, Boston, MA, USA, 2005.

[BS01] C. Baydar and K. Saitou. Prediction and diagnosis of propagated errors in assembly systems using virtual factories. *Application Brief, ASME Journal of Computers and Information Science in Engineering*, 1(3):261–265, 2001.

[BW01] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 2001.

[Cel91] F. E. Cellier. *Continuous System Modeling*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1991.

[CG08] J. Castro and R. Gavaldà. Towards feasible pac-learning of probabilistic deterministic finite automata. In *Proc. of the 9th intl. colloquium on Grammatical Inference: Algorithms and Applications*, ICGI '08, pages 163–174, Berlin, Heidelberg, 2008. Springer-Verlag.

[CGS10] M. P. Cabasino, A. Giua, and C. Seatzu. Fault detection for discrete event systems using petri nets with unobservable transitions. *Automatica*, 46(9):1531–1539, September 2010.

[CL08] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. 2.ed. Springer, 2008.

[CnCV93] M. A. Castaño, F. Casacuberta, and E. Vidal. Simulation of stochastic regular grammars through simple recurrent networks. In *Proc. of the Intl. Workshop on Artificial Neural Networks: New Trends in Neural Computation*, IWANN '93, pages 210–215, London, UK, 1993. Springer-Verlag.

[CO94] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *the Proc. of the Second Intl. Colloquium on Grammatical Inference and Applications ICGI '94*, pages 139–152, London, UK, 1994. Springer-Verlag.

[CO99] R. C. Carrasco and J. Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. In *RAIRO - Theoretical Informatics and Applications*, volume 33, pages 1–20, 1999.

[CT04] A. Clark and F. Thollard. Pac-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, December 2004.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[DA87] R. David and H. Alla. Continuous petri nets. In *Proc. of the 8th European Workshop on Application and Theory of Petri Nets*, pages 275–294. Zaragoza, Spain, 1987.

[DA01] R. David and H. Alla. On hybrid petri nets. *Discrete Event Dynamic Systems*, 11(1-2):9–40, January 2001.

[Dau92] I. Daubechies. *Ten lectures on wavelets*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[DD03] F. Desobry and M. Davy. Support vector-based online detection of abrupt changes. In *Proc. of the IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP'03)*, pages 872–875, Hong Kong, China, 2003.

[Dec97] S. E. Decatur. Pac learning with constant-partition classification noise and applications to decision tree induction. In *Proc. of the Fourteenth Intl. Conf. on Machine Learning*, ICML '97, pages 83–91, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[dKW87] J. de Kleer and B. C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[dlH97] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138, May 1997.

[dlH05] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.

[dlH06] C. de la Higuera. Data complexity issues in grammatical inference. In *Data Complexity in Pattern Recognition, Part I*, pages 153–169. Springer London, UK, 2006.

[dlH10] C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, 2010.

[dlHO04] C. de la Higuera and J. Oncina. Learning stochastic finite automata. In *Procs. of the 7th Intl. Colloquiuum on Grammatical Inference (ICGI), LNAI 3264*, pages 175–186, 2004.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[EAFT12] B. Esmael, A. Arnaout, R. K. Fruhwirth, and G. Thonhauser. Improving time series classification using hidden markov models. In *Proc. of the 12th Intl. Conf. on Hybrid Intelligent Systems HIS 2012*, pages 502–507, Pune, India, 2012.

[Eth08] Industrial communication networks - Fieldbus specifications - Part 5-10: Application layer service definition - Type 10 elements (IEC 61158-5-10:2007), 2008. DIN EN 61158-5-10:2008-09.

[Faw06] T. Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27:861–874, June 2006.

[FCTB10] D. P. Filev, R. B. Chinnam, F. Tseng, and P. Baruah. An industrial strength novelty detection framework for autonomous equipment monitoring and diagnostics. *Industrial Informatics, IEEE Transactions on*, 6(4):767 –779, nov. 2010.

[Fel50] W. Feller. *An introduction to probability theory and its applications*. John Wiley and Sons, New York, NY, USA, 1950.

[Fel11] M. Felser. *PROFIBUS Manual – A collection of information explaining PROFIBUS networks*. epubli GmbH, Berlin, Germany, 2011.

[FFP+12] S. Faltinski, H. Flatt, F. Pethig, B. Kroll, A. Vodenčarević, A. Maier, and O. Niggemann. Detecting anomalous energy consumptions in distributed manufacturing systems. In *Proc. of the 10th IEEE Intl. Conf. on Industrial Informatics INDIN'2012*, pages 358–363, Beijing, China, July 2012.

[GA08] M. S. Grewal and A. P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB*. John Wiley & Sons, Inc., Hoboken, New Jersey, USA, 2008.

[GB97] H. Guo and C. S. Burrus. Wavelets transform based fast approximate fourier transform. In *Proc. of the IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP '97)*, volume 3, pages 1973–1976, 1997.

[GCMC00] H. Guo, J. A. Crossman, Y. L. Murphey, and M. Coleman. Automotive signal diagnostics using wavelets and machine learning. *Vehicular Technology, IEEE Transactions on*, 49(5):1650 –1662, sep 2000.

[Ger98] J. J. Gertler. *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker Inc., New York, NY, USA, 1 edition, 1998.

[GJ90] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[GMY+07] R. Grosu, S. Mitra, P. Ye, E. Entcheva, I. V. Ramakrishnan, and S. A. Smolka. Learning cycle-linear hybrid automata for excitable cells. In *Proc. of HSCC07, the 10th Intl. Conf. on Hybrid Systems: Computation and Control, volume 4416 of LNCS*, pages 245–258. Springer Verlag, 2007.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[Gol78] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[Gro06] High-Level Group. MANUFUTURE - Strategic Research Agenda. Technical report, European Commission, 2006.

[Haw80] D. M. Hawkins. *Identification of outliers*. Chapman and Hall Ltd., London, UK; New York, NY, USA, 1980.

[HC01a] K. M. Hangos and I. T. Cameron. *Measurement and Instrumentation Principles*. Butterworth-Heinemann, Woburn, MA, USA, 2001.

[HC01b] K. M. Hangos and I. T. Cameron. *Process Modelling and Model Analysis*. Academic Press, London, UK, 2001.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proc. of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 278–292, Washington, DC, USA, 1996. IEEE Computer Society.

[Hen02] M. M. Henry. Model-based estimation of probabilistic hybrid automata. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge, MA, USA, May 2002.

[HKPV98] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.

[HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HT00] C. de la Higuera and F. Thollard. Identification in the limit with probability one of stochastic deterministic finite automata. In *Proc. of the 5th Intl. Colloquium on Grammatical Inference: Algorithms and Applications*, ICGI '00, pages 141–156, London, UK, 2000. Springer-Verlag.

[HTF08] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, New York, NY, USA, 2 edition, 2008.

[HW02a] M. W. Hofbaur and B. C. Williams. Mode estimation of probabilistic hybrid systems. In *Proc. of the Intl. Conf. on Hybrid Systems: Computation and Control*, pages 253–266. Springer Verlag, 2002.

[HW02b] M. W. Hofbaur and B. C. Williams. Proc. of the 13th intl. workshop on the principles of diagnosis (dx-02). pages 97–105, Semmering, Austria, may 2002.

[HZKW03] S. Hashtrudi Zad, R. H. Kwong, and W. M. Wonham. Fault diagnosis in discrete-event systems: framework and model reduction. *Automatic Control, IEEE Transactions on*, 48(7):1199 – 1212, 2003.

[IEC04] IEC/IEEE. Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEC 61588 First edition 2004-09; IEEE 1588*, 2004.

[Ise06] R. Isermann. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer-Verlag, Heidelberg/Berlin, Germany, 1 edition, 2006.

[JHZ12] S. Jabbari, R. C. Holte, and S. Zilles. Pac-learning with general class noise models. In *The 35th German Conf. on Artificial Intelligence (KI'2012)*, pages 73–84, 2012.

[JST09] JST. Failure knowledge database. *Japan Science and Technology Agency*, Online available at http://www.sozogaku.com/fkd/en/index.html, 2009.

[Kap98] J. N. Kapur. *Mathematical Modelling*. New Age International (P) Ltd., New Delhi, India, 1 edition, 1998.

[Kea98] M. Kearns. Efficient noise-tolerant learning from statistical queries. *J. ACM*, 45(6):983–1006, November 1998.

[KHH10] K. Kidam, M. Hurme, and M. H. Hassim. Technical analysis of accident in chemical process industry and lessons learnt. *Chemical Engineering Transactions*, 19:451–456, 2010.

[KMR+94] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *Proc. of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pages 273–282, New York, NY, USA, 1994. ACM.

[KNJ10] B. Kumar, O. Niggemann, and J. Jasperneite. Statistical models of network traffic. In *Intl. Conf. on Computer, Electrical and Systems Science,*, pages 146–154. Cape Town, South Africa, Jan 2010.

[KV94] M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, MA, USA, 1994.

[Lai88] P. D. Laird. *Learning from good and bad data*. Kluwer Academic Publishers, Norwell, MA, USA, 1988.

[Lap04] P. A. Laplante. *Real-Time Systems Design and Analysis: An Engineer's Handbook*. IEEE Press, Piscataway, NJ, USA, 3 edition, 2004.

[Lju99] L. Ljung. *System identification: theory for the user*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2 edition, 1999.

[Lyg06] J. Lygeros. Lecture notes on hybrid systems, AUT06-08. Technical report, December 2006.

[Man94] Siemens Manual. Teleperm xp, as 620 automation system, function block for group control level (ap), fb175 nlfilt (non-linear filter), 1994.

[Man06] Siemens Manual. Application on control technology - scaling and unscaling analog values v1.0, 2006.

[Man10] Siemens Manual. Sitrans p, series dsiii transmitters for pressure, differential pressure, flanged level, and absolute pressure model 7mf4*33-, 2010.

[MB98] P. J. Mosterman and G. Biswas. A theory of discontinuities in physical system models. *Journal of the Franklin Institute*, 335(3):401–439, January 1998.

[MB00] P. J. Mosterman and G. Biswas. Towards procedures for systematically deriving hybrid models of complex systems. In *Proc. of the 3rd Intl. Workshop on Hybrid Systems: Computation and Control*, HSCC '00, pages 324–337, London, UK, 2000. Springer-Verlag.

[MC04] H. Motulsky and A. Christopoulos. Fitting models to biological data using linear and nonlinear regression: A practical guide to curve fitting. Oxford University Press, New York, USA, 2004.

[MLD09] W. Mahnke, S. H. Leitner, and M. Damm. *OPC Unified Architecture*. Springer-Verlag Berlin Heidelberg, 2009.

[MNV+11] A. Maier, O. Niggemann, A. Vodenčarević, R. Just, and M. Jäger. Anomaly detection in production plants using timed automata. In *Proc. of the 8th Intl. Conf. on Informatics in Control, Automation and Robotics (ICINCO)*, pages 363–369, Noordwijkerhout, The Netherlands, July 2011.

[Moh09] R. Mohammadi. *Fault diagnosis of hybrid systems with applications to gas turbine engines*. PhD thesis, Concordia University, Montreal, Canada, 2009.

[Nar02] S. Narasimhan. *Model-based Diagnosis of Hybrid Systems*. PhD thesis, Vanderbilt University, Faculty of the Graduate School, Nashville, TN, USA, 2002.

[NB02] S. Narasimhan and G. Biswas. An approach to model-based diagnosis of hybrid systems. In *Proc. of the 5th Intl. Workshop on Hybrid Systems: Computation and Control*, HSCC '02, pages 308–322, London, UK, 2002. Springer-Verlag.

[NB07] S. Narasimhan and G. Biswas. Model-based diagnosis of hybrid systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 37(3):348 –361, May 2007.

[NM10] G. Nicolescu and P. J. Mosterman. *Model-Based Design for Embedded Systems*. CRC Press, Boca Raton, FL, USA, 2010.

[NMVJ11] O. Niggemann, A. Maier, A. Vodenčarević, and B. Jantscher. Fighting the modeling bottleneck - learning models for production plants. In *Proc. of the Model-Based Development of Embedded Systems Workshop MBEES2011*, pages 157–166. Dagstuhl, Germany, February 2011.

[NSV+12] O. Niggemann, B. Stein, A. Vodenčarević, A. Maier, and H. Kleine Büning. Learning behavior models for hybrid timed systems. In *Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, pages 1083–1090, Toronto, Ontario, Canada, 2012.

[OG92] J. Oncina and P. Garcia. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition, volume 5 of Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

[OS95] M. R. Osborne and G. K. Smyth. A modified prony algorithm for exponential function fitting. *SIAM Journal of Scientific Computing*, 16:119–138, 1995.

[OS09] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[Pit89] L. Pitt. Inductive inference, dfas, and computational complexity. In *Proc. of the Intl. Workshop on Analogical and Inductive Inference*, AII '89, pages 18–44, London, UK, 1989. Springer-Verlag.

[PKN+12] F. Pethig, B. Kroll, O. Niggemann, A. Maier, T. Tack, and M. Maag. A generic synchronized data acquisition solution for distributed automation systems. In *Proc. of the 17th IEEE Intl. Conf. on Emerging Technologies and Factory Automation ETFA'2012*, Krakow, Poland, September 2012.

[PM08] R. Pigan and M. Metter. *Automating with PROFINET: Industrial Communication Based on Industrial Ethernet*. Publicis Publishing, Erlangen, Germany, 2 edition, 2008.

[PN12] F. Pethig and O. Niggemann. A process data acquisition architecture for distributed industrial networks. In *Embedded World Conference 2012*, March 2012.

[PW93] L. Pitt and M. K. Warmuth. The minimum consistent dfa problem cannot be approximated within any polynomial. *J. ACM*, 40(1):95–142, January 1993.

[PW03] B. Peischl and F. Wotawa. Model-based diagnosis or reasoning from first principles. *IEEE Intelligent Systems*, 18(3):32–37, 2003.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Rab89] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–286, feb 1989.

[RAG04] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, Boston, London, UK, 2004.

[Reb67] A. S. Reber. Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior*, 6(6):855 – 863, 1967.

[Rei87] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[RN10] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, Upper Saddle River, NJ, USA, 3 edition, 2010.

[RST95] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *Proc. of the 8th annual conf. on Computational learning theory*, COLT '95, pages 31–40, New York, NY, USA, 1995. ACM.

[SC97] M. Sanseverino and F. Cascio. Model-based diagnosis for automotive repair. *IEEE Expert*, 12(6):33 –37, nov/dec 1997.

[Sch78] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. of the tenth annual ACM symposium on Theory of computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

[SE09] P. Struss and B. Ertl. Diagnosis of bottling plants - first success and challenges. In *Proc of the 20th Intl. Workshop on Principles of Diagnosis (DX-09)*, Stockholm, Sweden, 2009.

[Set10] B. Settles. Active learning literature survey. Technical report, January 2010.

[Slo95] R. H. Sloan. Four types of noise in data for pac learning. *Inf. Process. Lett.*, 54(3):157–162, May 1995.

[SLPQ06] P. Supavatanakul, J. Lunze, V. Puig, and J. Quevedo. Diagnosis of timed automata: Theory and application to the damadics actuator benchmark problem. *Control Engineering Practice*, 14(6):609–619, June 2006.

[TD97] Lj. Todorovski and S. Dzeroski. Declarative bias in equation discovery. In *Proc. of the 14th Intl. Conf. on Machine Learning*, ICML '97, pages 376–384, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[TDdlH00] F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic dfa inference using kullback-leibler divergence and minimality. In *Proc. 17th Intl. Conf. on Machine Learning*, pages 975–982. Morgan Kaufmann, 2000.

[Ton01] S. Tong. *Active Learning: Theory and Applications*. PhD thesis, Stanford University, Stanford, CA, USA, 2001.

[Tri02] S. Tripakis. Fault diagnosis for timed automata. In *Proc. of the Intl. Conf. on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'02)*, pages 205–224, 2002.

[UL07] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2007.

[Uv05] A. Ukil and R. Živanović. Detection of abrupt changes in power system fault analysis: A comparative study. In *Proc. of the Southern African University Power Engineering Conference*, Johannesburg, South Africa, 2005.

[Uv08] A. Ukil and R. Živanović. Adjusted haar wavelet for application in the power systems disturbance analysis. *Digit. Signal Process.*, 18:103–115, March 2008.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[VdWW07] S. Verwer, M. de Weerdt, and C. Witteveen. An algorithm for learning real-time automata. In *Benelearn*, pages 128–135, 2007.

[VdWW08] S. Verwer, M. de Weerdt, and C. Witteveen. Polynomial distinguishability of timed automata. In *Proc. of the 9th intl. colloquium on Grammatical Inference: Algorithms and Applications*, ICGI '08, pages 238–251, Berlin, Heidelberg, 2008. Springer-Verlag.

[VdWW09] S. Verwer, M. de Weerdt, and C. Witteveen. One-clock deterministic timed automata are efficiently identifiable in the limit. In *Proc. of the 3rd Intl. Conf. on Language and Automata Theory and Applications*, LATA '09, pages 740–751, Berlin, Heidelberg, 2009. Springer-Verlag.

[VdWW10] S. Verwer, M. de Weerdt, and C. Witteveen. A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In *Proc. of the 10th intl. colloquium on Grammatical inference: theoretical results and applications*, ICGI'10, pages 203–216, Berlin, Heidelberg, 2010. Springer-Verlag.

[VdWW12] S. Verwer, M. de Weerdt, and C. Witteveen. Efficiently identifying deterministic real-time automata from labeled data. *Machine Learning*, 86(3):295–333, March 2012.

[Ver10] S. Verwer. *Efficient Identification of Timed Automata: Theory and Practice*. PhD thesis, Delft University of Technology, Delft, the Netherlands, 2010.

[VKBNM11a] A. Vodenčarević, H. Kleine Büning, O. Niggemann, and A. Maier. Identifying behavior models for process plants. In *Proc. of the 16th IEEE Intl. Conf. on Emerging Technologies and Factory Automation ETFA'2011*, pages 937–944, Toulouse, France, September 2011.

[VKBNM11b] A. Vodenčarević, H. Kleine Büning, O. Niggemann, and A. Maier. Using behavior models for anomaly detection in hybrid systems. In *Proc. of the 23rd Intl. Symp. on Information, Communication and Automation Technologies-ICAT 2011*, Sarajevo, Bosnia and Herzegovina, October 2011.

[VMN13] A. Vodenčarević, A. Maier, and O. Niggemann. Evaluating learning algorithms for stochastic finite automata - comparative empirical analyses on learning models for technical systems. In *Proc. of the 2nd Intl. Conf. on Pattern Recognition Applications and Methods ICPRAM 2013*, pages 229–238, Barcelona, Spain, February 2013.

[Vod12] A. Vodenčarević. Learning behavior models of hybrid systems using wavelets for autonomous jumps detection. In *Proc. of the 10th IEEE Intl. Conf. on Industrial Informatics INDIN'2012*, pages 151–156, Beijing, China, July 2012.

[Vod13] A. Vodenčarević. Modelling abrupt changes: enhanced learning of behaviour models for manufacturing systems. *International Journal of Service and Computing Oriented Manufacturing*, 1(1):5–24, 2013.

[VTdlH+05a] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic Finite-State Machines-Part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1013–1025, 2005.

[VTdlH+05b] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite-state machines-part ii. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1026–1039, 2005.

[WD09] M. Wang and R. Dearden. Detecting and learning unknown fault states in hybrid diagnosis. In *Proc. of the 20th Intl. Workshop on Principles of Diagnosis (DX-09)*, pages 19–26, Stockholm, Sweden, 2009.

[Wei05] S. Weisberg. *Applied Linear Regression*. John Wiley & Sons, Inc., Hoboken, New Jersey, USA, 3rd edition, 2005.

[You82] S. J. Young. *Real Time Languages: Design and Development*. Ellis Horwood Publishers, Chichester, UK, 1982.

[ZKH+05] F. Zhao, X. D. Koutsoukos, H. W. Haussecker, J. Reich, and P. Cheung. Monitoring and fault diagnosis of hybrid systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(6):1225–1240, 2005.

[ZY01] H. Q. Zhang and Y. Yan. A wavelet-based approach to abrupt fault detection and diagnosis of sensors. *Instrumentation and Measurement, IEEE Transactions on*, 50(5):1389 –1396, oct 2001.