Identification of Timed Behavior Models for Diagnosis in Production Systems

Alexander Maier

Dissertation
in Computer Science

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, November 2014

I dedicate this thesis to my father.

Acknowledgements

Although many hours have been invested into this work, this work would not be possible without support of others.

First of all, I owe a great deal of gratitude to my supervisor Prof. Dr. Hans Kleine Büning for the support in scientific questions and for giving me the scientific freedom. Furthermore, I would like to give many thanks to Prof. Dr. Oliver Niggemann for many scientific discussions and for the good cooperation, which resulted in many common publications. I also give thanks to Dr. Meir Kalech for his willingness to review this thesis.

I would like to thank Dr. Asmir Vodenčarević for the good cooperation during both of our theses and for valuable discussions in research questions and writing publications together.

I thank my colleagues and students at the research institutes inIT and Fraunhofer IOSB-INA for the cooperative work: Thanks to Dr. Stefan Windmann for the project management, for integrating this work into the model identification and diagnosis framework proKNOWS. I thank Johann Badinger and David Schaffranek for the implementation of BUTLA and OTALA in the proKNWOS framework. Furthermore, I would like to thank the students Bernhard Jantscher and Fabian Böhm who implemented some other state-of-the-art algorithms (Alergia, MDI, RTI+) for evaluation purpose. I thank Björn Kroll, Florian Pethig and Darius Korzeniewski for implementing the anomaly detection and the data acquisition in proKNOWS, such that the algorithms can run in real production plant configurations. And I also thank Natalia Moriz for many discussions about mathematical problems.

Since English is not my native language, I thank Syed-Shiraz Gilani for proofreading the English grammar in this thesis and providing valuable suggestions.

I also give thanks to the German Federal Ministry of Education and Research (BMBF) for funding the project in which the dissertation was done.

I especially thank my parents for their immense support, I wish my father would still be here and could read this thesis.

But most of all, I thank my wife Erika and our sons Leon Patrick, Lukas and Julian for their immense patience and this work would not exist without their support.

Alexander Maier
Paderborn, November 2014

Abstract

Due to the increasing connectivity of automation devices and the increasing computing power of embedded devices (Cyber-Physical Systems), production plants are becoming increasingly complex. This leads to an increasing error susceptibility of the systems and thus to a severe fault detection and analysis.

The task of man is changing from operation to monitoring. But due to the increasing complexity of systems, operators are increasingly overwhelmed with the monitoring. The monitoring of the system essentially comprises the detection of faults in the regular production process.

Approaches from the field of model-based diagnosis are widely used for this task. Here, a model is used to predict the behavior of the system for given inputs. This prediction is compared with the actual behavior of the plant and in case of a deviation, an error is reported. Often, manually created models are used. But the manual creation of models is a time consuming and costly task. Thus, an automatic modeling is desirable.

This is where this work begins. The use of intelligent technical systems should support the operators in the monitoring of the system. In this work, three new algorithms are introduced, of which two algorithms deal with the identification of behavior models (in concretely: Timed Automata) and the third one uses the identified models for anomaly detection:

First, the algorithm BUTLA is introduced, which runs faster than other algorithms from state of the art. It is an offline identification algorithm, so it uses data which is stored in a database and a preprocessing of the data.

Then, the algorithm OTALA is introduced, which to the best of our knowledge is the first online passive identification algorithm for timed automata, that uses as few expert knowledge as possible. One of its benefits is the autonomous convergence detection. Therefore, this algorithm is especially suited for the usage in autonomously running Cyber-Physical Production Systems.

Finally, the anomaly detection algorithm ANODA is introduced. It uses the identified timed automata for anomaly detection.

The proposed algorithms are evaluated theoretically and empirically.

Zusammenfassung

Aufgrund zunehmender Vernetzung von Automatisierungsgeräten und der steigenden Rechenleistung von eingebetteten Geräten (Cyber-Physical Systems) werden Produktionsanlagen immer komplexer. Dies führt zu einer steigenden Fehleranfälligkeit der Systeme und damit zu einer erschwerten Fehler-Erkennung und -Analyse.

Die Aufgabe des Menschen wandelt sich vom Bedienen zum Überwachen. Aber aufgrund der steigenden Komplexität der Anlagen werden Maschinenbediener auch mit der Überwachung zunehmend überfordert. Die Überwachung der Anlage umfasst im Wesentlichen die Erkennung von Fehlern im regulären Produktionsablauf.

Weit verbreitet sind Ansätze aus dem Bereich der Modellbasierten Diagnose. Dabei wird ein Modell verwendet um das Verhalten der Anlage zu prognostizieren. Diese Prognose wird mit dem realen Verhalten verglichen und bei einer Abweichung ein Fehler gemeldet. Häufig kommen dabei manuell erstellte Modelle zum Einsatz. Die manuelle Erstellung der Modelle ist jedoch eine zeitintensive Aufgabe. Demnach ist eine automatische Modellerstellung wünschenswert.

Genau hier setzt diese Arbeit an. Der Einsatz von intelligenten technischen Systemen soll den Menschen in der Überwachung der Anlage unterstützen. In dieser Arbeit werden drei neue Algorithmen eingeführt, von denen zwei Algorithmen sich mit dem Lernen von Verhaltensmodellen in Form von zeitbehafteten endlichen Automaten befassen und der dritte die gelernten Automaten zur Anomalie-Erkennung verwendet:

Zuerst wird der Algorithmus BUTLA eingeführt, der schneller ausgeführt wird als andere Algorithmen (aus dem derzeitigen Stand der Technik). Es ist ein Offline-Lernalgorithmus, der auf Daten in einer Datenbank zugreift und eine Vorverarbeitung verwendet.

Dann wird der Algorithmus OTALA eingeführt, der nach unserem besten Wissen der erste passive Online-Lernalgorithmus für zeitbehaftete endliche Automaten ist und zudem mit minimalem Einsatz von Expertenwissen auskommt. Einer seiner Vorteile ist die automatische Konvergenz-Erkennung. Daher ist dieser Algorithmus speziell für die Verwendung in autonom laufenden eingebetteten Geräten geeignet.

Schließlich wird der Anomalie-Erkennungsalgorithmus ANODA eingeführt. Er nutzt die identifizierten zeitbehafteten endlichen Automaten zur Anomalie-Erkennung.

Die eingeführten Algorithmen werden theoretisch untersucht und empirisch ausgewertet.

Contents

Chapter 1

Introduction

1.1 Motivation

Modern production plants are becoming much more complex than just a few years ago. This is due to a high amount of automation in the production plants. As a consequence, the role of the human in the production industry is changing. Formerly, the role was to produce the products by hand. Then with first automated machines, the production became easier and faster. In today's modern production plants (at least with huge quantities), the role of the human is to ensure a smooth running of the automated production plant, i.e. the role of the human is changing from operating to monitoring. However, due to the continuously increasing complexity of the production plants, this task is also slowly becoming a tough task. Therefore, intelligent systems are required that assist the plant operator in this task.

Many approaches have been developed to support the plant operator in the monitoring task. A main focus is on the diagnosis in production plants. A possible solution is to create a model of the normal behavior and to use it during the regular plant operation to detect some anomal behavior (model-based diagnosis, MBD). The manual creation of behavior models is a tedious process and represents the bottleneck in the development of approaches for model-based diagnosis. For the modeling itself, much expertise is required and all interactions in the plant have to be known. Additionally, not all physical effects can be captured and modeled in detail.

It is desirable to create this kind of models automatically. The human is especially qualified to learn the normal behavior based on observations. This knowledge is then used during the running process to detect some anomal behavior.

As so often, we can learn from nature, so we try to identify behavior models automatically. To automate the model identification, approaches from the field of machine learning and artificial intelligence are used. Identifying behavior models from data is a complex problem in computer science. This is exactly the point where this thesis begins. In this dissertation, algorithms are developed that identify behavior models automatically from observations.

The automated identification of behavior models meets several major challenges:

Many approaches for model-identification assume the existence of a model structure and only learn the relevant parameters such as transition probabilities [SFJL03],

time ranges or coefficients of differential equations (e.g. [Ise04]). The identification of the entire model is still not the state of the art.

Another challenge is the trade-off between abstraction and accuracy (preciseness). On the one hand, it is desirable to keep the learned model as compact as possible. In most cases, it is not necessary to model all physical effects and their impact on the system behavior (black box). On the other hand, the model should contain all necessary information and the needed level of details to detect an abnormal behavior in the diagnosis phase.

A special focus in this thesis is on the factor of time. This aspect has not been sufficiently investigated so far in the current state of research. Existing methods often only learn the normal behavior of a system in terms of patterns in order to detect new behavior patterns in the diagnosis phase [MS03a, MS03b]. The identification of the timing behavior requires the consideration of other effects, such as the question whether a time deviation results from jitter or does it has to be classified as noise.

The model-identification generally requires a certain amount of a priori knowledge, for example knowledge about asynchronously operating subsystems. Although there exist some promising approaches, it is not yet possible to derive the causality reliably between the signals from the data alone. How much and which a priori knowledge is minimally necessary to allow a maximum degree of model-learning is an open research question in automation.

1.2 Relevance to Related Work

The identification of behavior models is an active research field. Behavior models can be learned as a neural network [CCW+11], trained as self-organizing maps [Fre08] or parametrized using a linear equation system or differential equations [Ise04].

In this thesis, we decided to use the formalism of finite state machines, more specifically, the timed automata. There already exist several algorithms for the identification of finite automata using observations. In general, there is a distinction between active and passive algorithms. Active algorithms allow to ask for new examples during runtime while passive algorithms have to deal with a given set of examples. The best known active algorithm is Angluin's L* [Ang87]. MDI [TDdlH00] and ALERGIA [CO99] are examples for passive algorithms which identify a probabilistic deterministic finite automaton. They use only positive examples, i.e. no failure measurements. Verwer already presented different algorithms for the identification of Timed Automata [Ver10]. Some of them use negative as well as positive learning examples. Furthermore, identification algorithms are distinguished identifying an automaton in an online or offline manner. To the best of our knowledge, there exists no algorithm for the online passive identification of timed automata.

The aforementioned algorithms mainly come from the field of grammatical inference and are therefore tested on artificial data only. There are only few examples, where a similar approach is used for the identification of behavior models in cyber-physical production systems. One approach is given in [Rot10]. They identify a Non-Deterministic Autonomous Automaton with Output (NDAAO), which not only considers the events and event sequences, but also the state information in form of a signal vector with inputs and outputs.

The goal of such identified models is the use for anomaly detection. Some approaches using statistical models are summarized in [MS03a, MS03b]. The behavior of a system is identified in form of patterns, which are then classified to distinguish between known behavior patterns (normal behavior) and unknown behavior patterns (faults, anomaly or novelty). However, the time factor is not considered. Since the behavior of cyber-physical production systems, or technical systems in general, always depend on time, other approaches have to be used. For this case, Timed Automata already have been used, examples are given in [LSS01, Tri02].

The algorithmic scope of this thesis is divided in two parts (see also Figure 1.1): Model identification and anomaly detection.

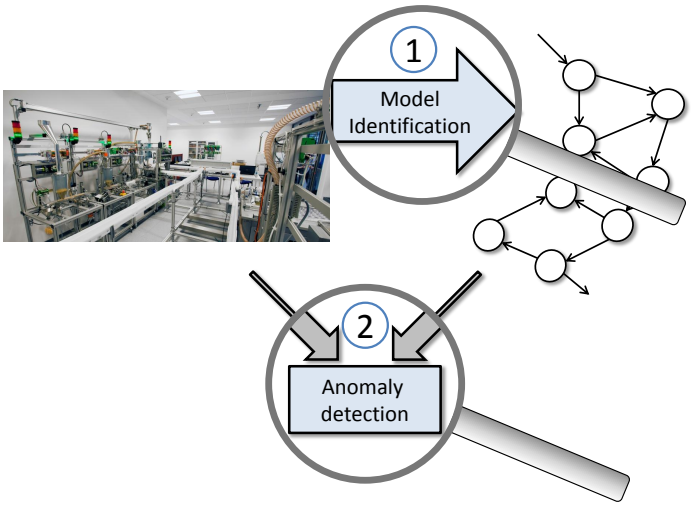


Fig. 1.1 The algorithmic focus is divided into two parts: Model identification and anomaly detection.

First, a new identification algorithm for Timed Automata (BUTLA) is introduced. It uses domain specific knowledge and extends the aforementioned algorithms using a new merging strategy, which results in a computation speed increase.

Focusing on the use case in cyber-physical production systems, some additional requirements have to be considered, for instance the ability to identify the model online. To the best of our knowledge, so far no online passive algorithm exists which is suitable for the learning of normal behavior in form of timed automata. For this purpose, the algorithm OTALA is introduced which closes the gap.

Additionally, a new anomaly detection algorithm ANODA is introduced, which uses the identified Timed Automata for the detection of error in the plant behavior.

The presented work was a joint research work with Asmir Vodenčarević. The focus of this thesis is on timing learning and development of the learning algorithm BUTLA, which identifies a Timed Automaton. The work in [Vod13] is based on BUTLA and it includes the continuous behavior. i.e. extending BUTLA to HyBUTLA, to identify a Hybrid Timed Automaton.

1.3 Contribution of this Thesis

The contributions of this work can be divided into two scopes: Algorithms and Theory.

1. **Algorithms**

   - First of all, we evaluated different modeling formalisms for the modeling of timing behavior and chose the most appropriate. The most important selection criterion is the ability for autonomous identification and for anomaly detection. The result of the evaluation is that Timed Automata are best suited for this task.
   - We developed the algorithm BUTLA for the identification of the timing behavior. This is the first automaton identification algorithm that uses the bottom-up merging strategy. The new merging order results in a computation speed increase compared to other algorithms using the top-down merging order.
   - We developed the algorithm OTALA, which is better suited for the learning of the normal behavior of cyber-physical production systems. This is the first online passive identification algorithm for Timed Automata. A big advantage of OTALA is that it recognizes the convergence of the identification process and is therefore able to autonomously abort the identification process at the earliest point in time.
   - We developed the algorithm ANODA for the anomaly detection in production plants. It uses the automata of both BUTLA and OTALA. Using this algorithms, logical, timed and probabilistic error can be detected.
   - The developed algorithms have been tested in real-world applications. The identified models were used for anomaly detection. The algorithms were nested in a model identification and anomaly detection framework. This has been applied to a model factory at the research institute and in a process plant at Jowat AG.

2. **Theory**

   - We prove that the bottom-up strategy performs faster on average than the top-down strategy.
   - We prove that BUTLA identifies Timed Automata weakly in the limit. The runtime of BUTLA is polynomial to the input size.
   - We prove that, under certain conditions, BUTLA identifies Timed Automata strongly in the limit. Under the given conditions, BUTLA only uses learning examples whose amount is polynomial to the size of the final automaton.
   - We prove that, under certain conditions, BUTLA identifies the timing behavior in the limit. Given enough learning examples, the identified timing behavior corresponds to the correct timing behavior.
   - We prove that, despite the exponential worst case runtime, OTALA can still be used for the identification of Timed Automata in cyber-physical production systems.
   - We prove that $ANODA \in \mathcal{L}$, i.e. the anomaly detection using Timed Automata uses space logarithmic to the input size and therefore can be performed efficiently.

1.4 Realized and Potential Application Scenarios

The automatic model identification has several potential applications. Most of them have already been realized in our working group at the institute. Many works arose from this thesis.

Model-based anomaly detection:

The model-based anomaly detection is the most obvious potential application. It is also a key point in this thesis. The principle is the following: A model is used during runtime and the behavior of the model is compared with the behavior of the running plant. If some deviation arises between the prognosis of the model and the current plant behavior, an anomaly is signaled.

Some work has already been done in the field of model-based anomaly detection. However, most of them rely on a model that has been manually created. In this thesis, we propose an approach in which the model is identified automatically, i.e. without using expert knowledge. The introduced algorithms close the gap of manual model creation for timed technical systems.

Model-based diagnosis goes a step deeper. It includes the anomaly detection and the detection of the error cause. In our work so far, we only focused on model-based anomaly detection. We captured the scope of model-based anomaly detection in several publications [MNV+11, NSV+12, MKPGN13, KSSN14, KAB14] .

Model-based design:

The model-based design allows the user an early test of the components to be developed. Using behavior models of the components, selected modules can be specifically checked on selected properties. Furthermore, it offers the possibility to check the fulfillment of specifications by simulation. For instance, some device model is used to simulate the network behavior in the early development phase. In [GKN+11], we described a concept of how such a simulation is possible: A component-based simulation framework uses individual simulation components and combines them to form a complete model. AutomationML is used for the signal mapping between the PLC program and the simulation components. This allows building a topology model; or different topology models whose properties can be examined.

Furthermore, in [MKPGN13], we described an approach that how the algorithms proposed in this thesis can be used to identify device models (behavior of single network components like IO-devices) and how these models can be used for model-based design.

Model-based testing:

Model-based testing approaches can be used to automate test activities or to generate test case for the test process. For this, different approaches exist. However, they all need models. The models can be created manually or, as we propose in this thesis, generated automatically. E.g. in [SMK+13] we proposed a method based on Hardware-In-The-Loop (HIL). Other approaches (also developed at our research intitute) can e.g. be found in [KTN+12, KNSJ12, KDSJ14]

Plug & produce:

Another use case for models in cyber-physical production systems is plug & produce. The goal is that the production systems and their automation systems automatically connect to each other and can be adapted automatically to new

scenarios such as new product variants to change the production modules. In order to do so, the knowledge about the behavior of single components is required. This is where the (automatically identified) models come into the game. In [OHN14], an approach is proposed that uses our proposed model identification algorithm to obtain a first behavior model based on generated event sequences. This behavior model is adapted afterwards, based on real measurements.

Optimization:

The automatically identified models can also be used for optimization. Using Hybrid Automata as modeling formalism for optimization and mode-predictive control (e.g. [BC11]).

A possible application scenario could be the minimization of energy consumption. The hybrid automaton holds the information about the energy consumption (it is not more than a continuous signal) for each identified state. This gives an advantage compared to a global power meter, which only holds information about the overall energy consumption. Since the energy consumption is known for each state, the cause for high energy consumption can be located more specifically.

1.5 Overview

This thesis is organized in four parts: *(1)* Foundations, *(2)* Algorithms and Theory, *(3)* Applications and *(4)* Conclusion. The chapters are organized as follows:

Part I Foundations

The first part gives an overview to the foundations and the state of the art. Known algorithms for the identification of finite automata are described and the most important known results from the complexity analysis are summarized.

Chapter 2 analyzes the problem of identifying the normal behavior of timed systems. Relevant modeling formalisms are compared regarding the special criteria. Additionally, the challenge of identifying the timed behavior is analyzed and finally, the choice of Timed Automata as modeling formalism is argued.

Chapter 3 leads into the topic by introducing the used terms and formalisms formally.

Chapter 4 reviews the state of the art concerning the complexity of identification of finite automata.

Chapter 5 reviews the state of the art concerning the existing algorithms for model identification (especially finite automata) and existing diagnosis approaches. Additionally, in the conclusion, the research gap is pointed out. Part II is aiming at closing this gap.

Part II Algorithms and Theory

The second part is the core of this thesis. It introduces the algorithms and evaluates them while examining theoretical and empirical aspects.

Chapter 6 introduces the model identification algorithms BUTLA and OTALA and the anomaly detection algorithm ANODA.

Chapter 7 evaluates the introduced algorithms regarding theoretical aspects, like runtime or correctness.

Chapter 8 uses artificial data to evaluate the algorithms empirically.

Part III Applications

The third part (contains only *Chapter 9*) gives the proof of concept by describing the realized application scenarios, which are particularly a model identification framework and two real-world scenarios in the production and process industry.

Part IV Conclusion

In *Chapter 10* finally, the thesis is summarized and concluded. Additionally, an outlook to future work is given.

Part I

Foundations

Chapter 2

The Core Issue: Modeling and Learning Time

This chapter places the core issue of this thesis into the scientific landscape of time modeling and time learning.

The modeling of time for computation purpose is a widely researched area. Many formalisms have been created to model different aspects of timing behavior. In this chapter, some aspects are analyzed which have to be considered when choosing an appropriate timing modeling formalism. Based on this analysis, some modeling formalisms are evaluated according to their capabilities to model the timing behavior. One of those formalisms is chosen that is well suited for the application of this thesis: the anomaly detection in cyber-physical production systems. For this, the formalisms are introduced informally, formal definitions and more detailed information can be found in the referred literature. The formalisms which are used later on in this thesis are defined formally in Chapter 3.

To keep the application domain in mind, a special focus is on modeling and identification of the timing behavior of cyber-physical production systems. Additionally, the suitability of the modeling formalisms according to automatic learnability from observations only and the suitability for anomaly detection is evaluated.

The contributions given in this chapter are the following:

- We classify the timing modeling problem according to specified aspects in a taxonomy.
- We analyze the modeling formalisms according to the suitability for automatic learning.
- We choose an appropriate modeling formalism and reason the usability for the desired application.

In Section 2.1, the most important aspects for time modeling are listed and evaluated in a taxonomy. Section 2.2 analyzes the possibilities that how timing can be learned based on different modeling formalisms. Finally in Section 2.3, the choice of an appropriate formalism for the aforementioned purpose (identification and anomaly detection) is argued and the challenges of the problem of timing learning are pointed out.

2.1 Modeling Time - A Taxonomy

In [FMMR10], Furia et. al. give a comparative survey on time modeling formalisms and summarize the most important features into a taxonomy. This section is mostly based on that survey. The features of the taxonomy are analyzed with respect to the usage of formalisms for the modeling of timed production systems and supplemented with other features.

Discrete vs. Dense Time Domains

The separation of formalisms concerning the usage of discrete and dense time domains is a first natural categorization. Discrete time models comprise a set of isolated points, whereas dense time means that in a dense set, ordered by "<", for every 2 points t_1 and t_2 with $t_1 < t_2$ there is always a third point t_3 in between, such that $t_1 < t_3 < t_2$.

More detailed distinctions are useful to better evaluate the timing model formalisms:

- *Finite or bounded time models:* Often modeled timed systems consider time to be infinite. The behavior can proceed indefinitely in the future and also in the past. Therefore, it is natural to model time as an unbounded set. However in many cases, a bounded time frame is necessary.
  As an example, the filling of a container is considered. The process is time bounded, both in the past (starting of the filling process) and in the future (stopping the filling process or container is completely filled. If the container is filled within maximum 10 seconds, a time range [0 sec...10 sec] is given. Without loss of generality, the model for filling the container can be analyzed only within this bounded time range.

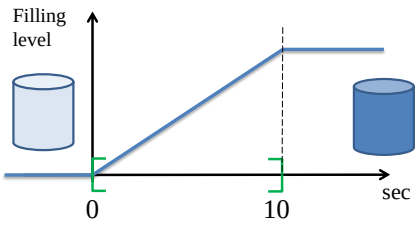


Fig. 2.1 Bounded time model: Filling of a container.

  Another example is the modeling of periodic signals. Without the loss of generality, only one cycle can be modeled to obtain the knowledge of the complete system.
- *Hybrid systems:* Timed systems can also be hybrid. In this case, hybrid means that a model uses discrete as well as dense domains. Several reasons lead to the necessity of hybrid models:
  - A system with a discrete and maybe finite set of states is modeled using a dense time domain. In a graphical representation, this leads to a square wave form and the states are displayed as piecewise constant functions over dense time.

– In a similar way, a continuous behavior can be sampled with a regular interval by using a sampler.
– Another case which leads to a hybrid model arises when discrete events are modeled combined with continuous variables. This is especially the case in timed and hybrid automata.

Explicit vs. implicit modeling of time

Another major distinctive feature is the possibility of implicit and explicit modeling of time. Model formalisms with explicit time allow the modeling of concrete time values for some specific event, e.g. "if the sensor is activated, start the conveyor belt within two seconds". Implicit time modeling only gives information about the time duration as a whole, e.g. the durations of states or actions, see Figure 2.2.

ODEs, as an example, only give the possibility of implicit time modeling. Other formalisms, mainly state-based formalisms such as statecharts, Petri nets and timed automata allow both implicit and explicit time modeling. Timed automata, for instance, have a relative time stamp for events. This time stamp is an explicit time value. Adding all relative time stamps leads to an absolute time value based on the beginning of the process, this is an implicit time value [FMMR10].

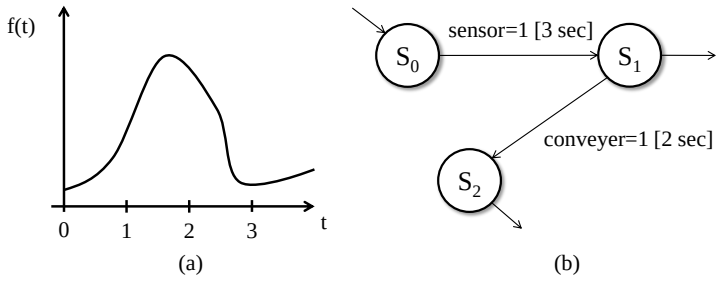


Fig. 2.2 Implicit (a) and explicit time model (b).

One clock vs. many clocks

Furthermore, time model formalisms can be differentiated according to their number of used clocks. When dealing with independent modules within a system, the question arises whether to use one or many clocks. The usage of many clocks leads to the need of clock synchronization in the simulation step, whereas the usage of one clock only requires a transformation from an n-clock model to a 1-clock model.

Timed Automata allow both, one and many clocks. However, in [Ver10] Verwer showed that 1-clock-Timed Automata and n-clock-Timed Automata are language-equivalent, but in contrast to n-clock-Timed Automata, 1-clock-Timed Automata can be identified efficiently.

Concurrency and composition

Most real systems are too complex to model them in one overall model. The behavior has to be divided into several subsystems, so that the overall model is a composition of its sub-models. For finite state machines, we will see in section 3.1

that the number of states reduces enormously if the system is decomposed into subsystems. This is also referred to as modularization.

The decomposition is a less mature process. Difficulties can arise in the synchronization step. Mostly, the separated models of subsystems have equal or identical properties. Furthermore, the time bases can be different between the modules, discrete or continuous, or the time base is implicit for one module and explicit for another.

Single-mode and multiple-modes

The distinction between models, which can only cope with single-modes and models that additionally can deal with multiple-modes, goes a step deeper than concurrency and decomposition. A system may, at some point in time, abruptly change its behavior. In technical systems, this happens for reasons such as shifting a gear or stopping a conveyor belt. All state based models (e.g. statecharts, Petri nets or finite state machines) are able to describe multiple-mode systems, where equation based formalisms (e.g. ordinary differential equation) can only describe the behavior of single-mode systems. Figure 2.3 shows a simplified example for a multi-mode model for shifting a gear. The figure in 2.3 (a) tries to capture the behavior in a diagram. This behavior cannot be described using one differential equation. The figure in 2.3 (b) shows the same behavior in a hybrid automaton, which is (at least for this example) much more elegant.

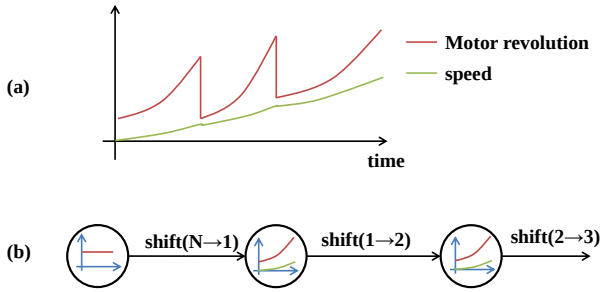


Fig. 2.3 Multi-mode model in a (a) diagram and (b) hybrid automaton.

Linear- and branching-time models

It can also be differentiated between linear and branching time models [Var01]. Linear-time formalisms are interpreted over linear sequences of states. Each description refers to (a set of) linear behaviors, where the future behavior from a given state at a given time is always identical. Branching-time formalisms are interpreted over trees of states. That means, in contrast to linear-time models, the future behavior of a given state at a given time can follow different behavior according to the tree (see also Figure 2.4).

A linear behavior can be regarded as a special case of a tree. Conversely, a tree can be treated as a set of linear behaviors that share common prefixes (i.e., that are prefix-closed); this notion is captured formally by the notion of fusion closure [AH92]. Thus, linear and branching models can be put on a common ground and compared.

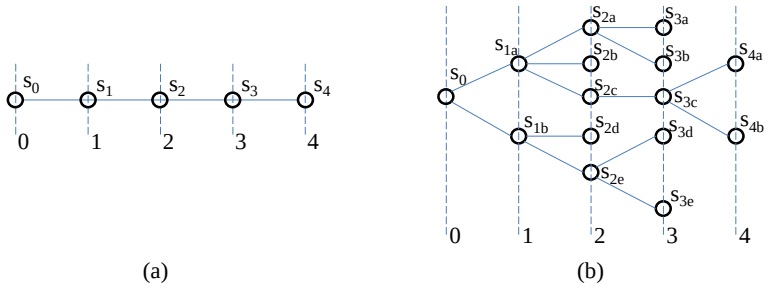


Fig. 2.4 A linear (a) and branching time model (b) [FMMR10].

It is also possible to have semantic structures which were branching in the past, which means that different pasts merge into one single present [Koy91]. Branching-in-the-past models are very rarely encountered in practice.

Linear time models, in general, work in a deterministic manner. Whenever an input is read, the system behaves uniquely, i.e. the outputs are set deterministically. In contrast to linear time models, branching time models are non-deterministic, since the system can "choose" which state to go next.

2.2 Learning Time - The Challenge

Section 2.1 informally introduced several time modeling features and some formalisms that satisfy certain criteria. However, as indicated in the introduction, it is not intended to create timing models manually. Instead, the models should be identified automatically. This section reviews the applicability of some modeling formalisms with respect to automatic learning and the suitability for anomaly detection.

The evaluation of these formalisms concerning learnability is intended to answer the following questions:

- Which method can learn which effects?
- Which modeling formalism can be used?
- What are the possibilities of certain formalisms?
- What are the limits?
- Can the models be used for anomaly detection?

This section considers the learnability of timed systems concerning several formalisms. In Subsection 2.3.2 the challenges in learning Timed Automata are outlined. A more detailed analysis follows in Chapter 4.

For further analysis, we divide the considered formalisms into *(1) Dynamic system models* (Section 2.2.1), *(2) operational formalisms* (Section 2.2.2) and *(3) descriptive formalisms* (Section 2.2.3). In the following subsections these formalisms are analyzed concerning their capabilities of modeling technical systems and whether such kinds of models can be identified automatically. For each group at least one modeling formalism is evaluated in more detail.

2.2.1 Dynamic system models

In various engineering disciplines (like mechanical or electrical) and especially in control engineering, the so-called *state-space representation* is a common way to model the timing behavior of technical systems [Kha02].

Three key elements are essential for the state-based representation: The vector $\mathbf{x}$ with the state variables, the vector $\mathbf{u}$ with the input variables and the vector $\mathbf{y}$ with the output variables. All these values explicitly depend on the time at which they are evaluated (usually represented as $\mathbf{x}(t)$, $\mathbf{u}(t)$, and $\mathbf{y}(t)$), i.e. it uses explicit timing (see Section 2.1).

The temporal domain in state-based representation is either continuous (e.g. $\mathbf{R}$) or discrete (e.g. $\mathbf{Z}$). Using the continuous temporal domain, the relationship between $\mathbf{x}$ and $\mathbf{u}$ can be expressed through differential equations:

$$\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t), t), t \in \mathbf{R} \tag{2.1}$$

Using the discrete temporal domain, the relationship between $\mathbf{x}$ and $\mathbf{u}$ can be expressed through difference equations:

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k), k), k \in \mathbf{Z} \tag{2.2}$$

Based on an initial condition $\mathbf{x}(0)$ and given input $\mathbf{u}(t)$, all calculations for $\mathbf{x}(t)$ of equation 2.1 and for $\mathbf{x}(k)$ of equation 2.2 represent the system behavior. The same combination of initial values and input values always results in the same solutions, i.e. the systems that are modeled using state space representation are usually deterministic. Thus, dynamical system models usually presuppose a linear time model.

The main advantage of dynamical system models is that very detailed physical models can be created using established mathematical methods. But this also can turn into a disadvantage. For many purposes, the models are too detailed, i.e. they are unsuitable for high-level description, since some expert knowledge is required to read and understand the models. Furthermore, dynamical system models are unsuitable for the modeling of distributed systems.

Various methods exist to identify dynamic system models. These methods are grouped under the term model identification (sometimes the term "system identification" is also used). In [IM10], Isermann describes some methods, e.g. by means of parameter estimation. The states themselves are not identified. A structure model is presumed and the identification methods determine the parameters. So, still some expert knowledge is necessary and manual work has to be done.

Dynamical system models also can be used for fault detection (e.g. [Ise04]. Figure 2.5 shows the general scheme of process model-based fault detection using dynamical system model according to [Ise04]. The model-based fault detection uses the inputs $\mathbf{u}$ and the outputs $\mathbf{y}$ to generate residuals $\mathbf{r}$, the parameter estimates $\mathbf{\Phi}$ or state estimates $\mathbf{x}$, that are called features. A comparison of these features with the nominal values (normal behavior) detects changes of features, which lead to analytical symptoms s. The symptoms are then used to determine the faults.

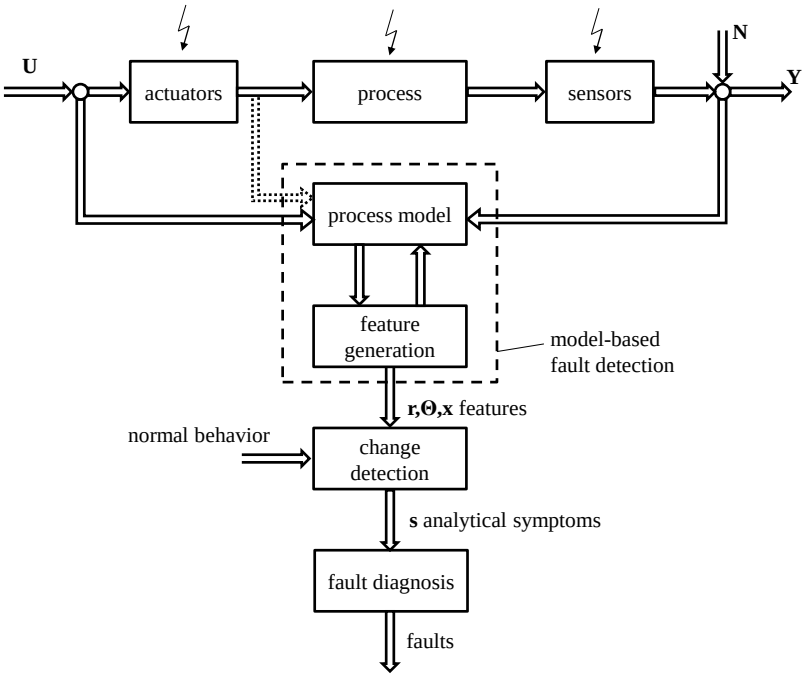


Fig. 2.5 General scheme of process model-based fault detection using dynamical system model according to [Ise04].

2.2.2 Operational Formalisms

Operational formalisms can be roughly divided into synchronous state machines and asynchronous abstract machines. These types are described in more detail in this section.

Synchronous state machines

A large variety of synchronous state machines exists: finite state machine, statecharts, timed automaton, hybrid automaton, Büchi automaton, Muller automaton, and others (see [Tho90]). In this section we confine our self to finite state machines and timed automata, the timing extension of finite state machines.

Finite state machines mainly consist of states and transitions between the states. The transitions are fired if the condition is fulfilled. As Figure 2.6 shows, the first transition is fired if an item is recognized at position A. This is also called a (discrete) event. The behavior of a system is described as a computation path, i.e. one run through the automaton from the initial state to the finite state.

The main strength and the reason for the wide usage of finite state machines is their accessibility for humans and their simplicity. Often, processes or timing behavior are described by a sequence of events. In fact, technical systems are often programmed in state machines, e.g. using the standardized programming language from IEC 61131. Therefore, modeling the timing behavior of such technical systems, in the sense of finite state machines or timed automata, is consequential. Figure 2.6 shows a simple example: If a bottle is placed on position A, the conveyer

belt is switched on. After reaching the position B, the conveyer belt is switched off again.

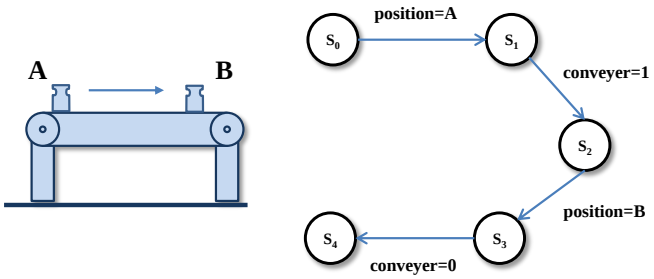


Fig. 2.6 A technical system and corresponding state machine.

Another main advantage is the ability of abstractness. In most cases, the model does not need all the details that could be modeled. The goal is mainly to compute the timing behavior of systems (with focus on time), while the computation of the exact physical behavior is less important. In particular, from the point of view of the computer scientist, they are especially appealing, as they allow to reason about time in a highly simplified way.

Finite state machines allow no explicit modeling of time. However, using variations like timed automata, both explicit and implicit time modeling is possible.

Due to the simplicity, finite state machines are further usable for automatic verification [CGP99].

Usually automata are non-deterministic since they comprise multiple computation paths (compare "branching time model" in Section 2.1). However, this can also be deterministic (compare "linear time model" in Section 2.1), where multiple runs of the automaton represent the possible behavior in general. This is mostly more intuitive for the user.

In contrast to asynchronous abstract machines (e.g. Petri nets, see next paragraph), finite state machines have no possibility to model concurrency. If it is desired to model concurrent behavior of parallel running processes, one finite state machine for each process has to be created and afterwards all parallel finite state machines are joined together. The most common concurrency model is *synchronous* concurrency. It uses a global time base for all parallel finite state machines and concurrent transitions that occur simultaneously. Further details on concurrent finite state machines can be found in [LL98] and [GLL99].

Different types of automata have different features. Based on the basic finite state machine, a wide range of sub types exists (probabilistic, non-deterministic, timed, hybrid and other). A more detailed overview and formal definitions to the variety of finite state machine formalisms will follow in Section 3.2.

Another very important issue in the context of this thesis is the applicability of the formalism for automatic identification. As mentioned before, technical systems are often programmed based on finite state machines. From this, it follows that the state machines (including its sub classes) can be identified by observations.

Several algorithms exist to learn finite state machines. A detailed overview follows in Section 5.2

Finally, finite state machines can also be used for fault detection and diagnosis (e.g. in [Tri02, SFJL03, SADMK07]). For diagnosis, the learned automata can be used as well as manually created ones. Depending on the used formalism, different errors can be detected: wrong event sequence, improper event, timing deviation and error in continuous signals.

Asynchronous abstract machines

Beside the finite state machines, which work synchronously, there exist formalisms that work asynchronously, called the asynchronous abstract machines. The most popular formalism in this group is Petri nets.

Petri nets are named according to Carl Adam Petri, who initially developed this modeling formalism [Pet62]. A variety of Petri nets exists. The most common type is Place/transition-nets. It basically consists of states and transitions. Places store tokens and hand them over to the transitions. If all incoming places hold at least one token, a transition is enabled. An enabled transition can be fired. After firing the transition, the tokens from the incoming transitions are moved to the outgoing transitions. An example of a Petri Net with two execution steps is illustrated in Figure 2.7.

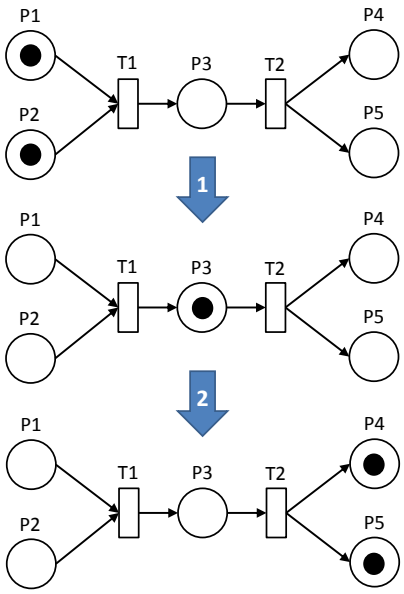


Fig. 2.7 An example Petri net with two execution steps.

It can easily be seen that FSMs can be transformed into Petri nets. Basically, the Petri Net places correspond to states in the FSM, Petri Net transitions directly correlate to transitions in FSMs, the starting state corresponds to a place holding a token at the beginning and each final state in the FSM generates a transition in the Petri Net, which throws a token away.

Petri nets also have been extended to handle timing information. Merlin and Farber proposed the first Timed Petri net in [MF76]. Each transition is extended with the minimum and maximum firing time, where the minimum firing time can be 0 and the maximum can be ∞. A comprehensive survey on several timed extensions to Petri nets can be found in [Cer93] and [CMS99].

Furthermore, several approaches exist to identify Petri nets from sampled data. However, some requirements are put on the language to be identified or some assumptions are made, e.g. in [CGS07], Petri nets are identified from knowledge of their language, where it is assumed that the set of transitions and the number of places is known. Only the net structure and the initial marking are identified.

Petri nets in general are suited for fault detection (e.g. in [NDZ13] or [MWS13]). The different types of Petri nets (mainly condition/event-systems, Place/transition-nets and High-level Petri nets) have different time and space complexity. This will be analyzed in more detail in Section 2.3.1.

2.2.3 Descriptive Formalisms

As the name suggests, descriptive formalisms describe the model using a natural language, mostly based on mathematical logic [BCM$^+$03]. Such formalisms are especially suited if some conditions have to be described.

Example 1. *If it is raining or if it was raining in the last two hours, then the street is wet.*

Similar rules can also be created for the prediction of output signals (actuators) based on the inputs (sensors) in a cyber-physical production system.

As already shown in the example, the conditions can also contain time information.

There exist different types of descriptive formalisms, e.g. first order logics, temporal logics, explicit-time logics or algebraic formalisms. Further details can be found in the literature, e.g. [BCM$^+$03].

Some algorithms exist to identify descriptive models. For the prediction of the behavior of cyber-physical production systems, a timed decision tree can be learned for instance.

Examples for such learning algorithms are ID3 [Qui90], the C4.5 algorithm as extension of the ID3 algorithm [Qui93] or a generic algorithm for building a decision tree by Console [CPD11].

Note that the rule can not always be interpreted backwards. Using the last example a reason for the wet street could be that somebody has washed his car on the street. Therefore, descriptive formalisms have a limited suitability for anomaly detection. The usage of descriptive formalisms for anomaly detection puts additional requirements on the rules, they have to be more concrete. Using the given example, it can be modified as follows:

Example 2. *The street is wet if and only if it is raining or it was raining in the last two hours.*

This rule allows a backward interpretation if the reason for the wet street is unknown. However, the meaning of the rule has now changed.

2.3 Conclusion

In this thesis we focus on the modeling formalism of Timed Automata. This Section gives some reasons for why Timed Automata were chosen as modeling formalism and what the challenges are in learning Timed Automata from observations only.

2.3.1 The choice of formalism to use

We have seen that many formalisms can be used to model the timing behavior. We decided to use Timed Automata for the following reasons (the term *finite state machines* can be also replaced by *timed automaton*):

- **Intuition:** Finite state machines are very intuitive for humans. They are easy to understand.
- **Understandability:** In contrast to many other automatically identified models, the identified finite state machines are understandable for humans. They can be verified by experts.
- **Wide usage:** Finite state machines are widely used, e.g. for modeling or programming.
- **Learnability:** Finite state machines are suitable for automatic learning.
- **Diagnosability:** Finite state machines are suitable for fault detection. This applies for both, manually created and automatically identified finite state machines.
- **Suitability for verification:** The identified finite state machines can be used for automatic verification.
- **Modification:** The identified finite state machine can be manually modified and adapted after learning. This can also be done automatically.

However, it may be held that these arguments are also true for other formalisms, such as Petri nets, Hidden Markov Models or Push-down automata. To refine the selection, we recall the actual application, that uses models for anomaly detection.

The anomaly detection in cyber-physical production systems takes the current observation and checks whether it belongs to the normal behavior, i.e. the model of the normal behavior. In theoretical computer science, there exists a similar problem, the word problem, in which a word is given and it is checked whether it belongs to a certain language. Therefore, from the theoretical computer science point of view, the anomaly detection problem corresponds to solving the word problem. So, we can use the Chomsky hierarchy to refine the selection of possible modeling formalisms. Table 2.1 shows the four levels of the Chomsky hierarchy and the corresponding time and space complexity for solving the word problem.

The word problem for regular languages can be solved in linear time on constant space, whereas the time and space complexity is worse for context-free and context-sensitive languages. For type-0 languages, the word problem is not even decidable. Applying this to the anomaly detection problem, it is evident that only regular languages can be used due to the linear time complexity. Context-free and context-sensitive languages cannot be used since it is necessary to have the complete word to solve the word problem (e.g. the CYK algorithm, which is named after its inventors Cocke Younger and Kasami, to solve the problem for context-free languages, which additionally requires that the grammar is given in Chomsky normal form). This can

Table 2.1 Time and space complexity (with n as the input size) for solving the word problem according to Chomsky hierarchy.

languages	**time complexity**	**space complexity**
regular (type-3)	$\mathcal{O}(n)$	$\mathcal{O}(1)$
context-free (type-2)	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
context-sensitive (type-1)	$\mathcal{O}(2^n)$	$\mathcal{O}(n)$
(type-0)	not decidable	

not be used for the application in anomaly detection, since an error should be signaled here as soon as it appeared and not after finishing the production cycle.

As a consequence, we can state that we need a modeling formalism which can describe regular languages (type-3 in the Chomsky hierarchy). Table 2.2 compares FSMs with Petri nets using the classification according to Chomsky hierarchy. It can be seen that push-down automata and the "classical" Petri nets (the Place/transition-nets) cannot be used together with Turing machines. Finite automata and condition/event-systems, which is a restricted subclass of Petri nets, are classified in level three. Condition/event-systems have three limitations: (i) only one token per place is allowed, (ii) capacity of places is limited to one token, (iii) edge weights are limited to 1. With these limitations, the formalism of condition/event-systems compared to finite automata only differs in the point of concurrency. However, it was already mentioned that concurrency is very difficult to identify. In our approach, the information about systems running in parallel (concurrency) is assumed to be given as expert knowledge and the behavior is identified as finite automaton.

Table 2.2 Comparison of state machines and Petri nets and the classification according to Chomsky hierarchy.

languages	**machines**	**Petri nets**
regular (type-3)	finite automata	condition/event-systems
context-free (type-2)	push-down automata	Place/transition-nets (neither $L_{PN} \subseteq L_{CF}$ nor $L_{CF} \subseteq L_{PN}$)
context-sensitive (type-1)	linear-bounded Turing machines	
(type-0)	Turing machines	High-level Petri nets

Furthermore, the question that may arise is why the timing behavior is not learned using the formalism of Markov chains. Usually, Markov chain identification algorithms assume the structure (states and transitions) as given and only the transition probabilities are identified. As an example, the speech recognition is mentioned. A Markov chain gives the probabilities of all combinations of subsequent syllables, i.e. we obtain a fully meshed graph. In the domain of cyber physical production systems a fully meshed graph is an absolutely rare case (or even impossible). Therefore, we decided to use finite state machines. However, finite state machines are a special case of Markov chains where the markov property is set to 1.

2.3.2 The challenges in Learning Timed Automata

In the previous sections we have argued that timed automata are best suited to model the behavior of timed systems because of the simplicity and level of abstractness. Furthermore, they are best suited to learn the behavior model automatically. However, there are still some challenges in learning timed automata. These are shortly summarized here. Details will follow in Section 4.

- **Identification of states and events:** The timing behavior includes not only the time stamps for some observations, but also some states and transitions with timed events in between. Many learning algorithms (especially for learning of Markov Chains) assume the states and transitions as given and only learn the transition probabilities. Here, the structure (states and events) is not given but has to be identified from observations.
- **Number of clocks:** Technical systems may be programmed using a certain number of clocks. These have to be identified or the behavior has to be expressed using only one clock.
- **Event splitting:** When do events with different timing belong to the same event; or do they describe different events? As can be seen in Figure 2.8, the events can be split based on the timing, which is based on the container size. More formally: The event's timing distribution function can comprise several modes that have to be identified.

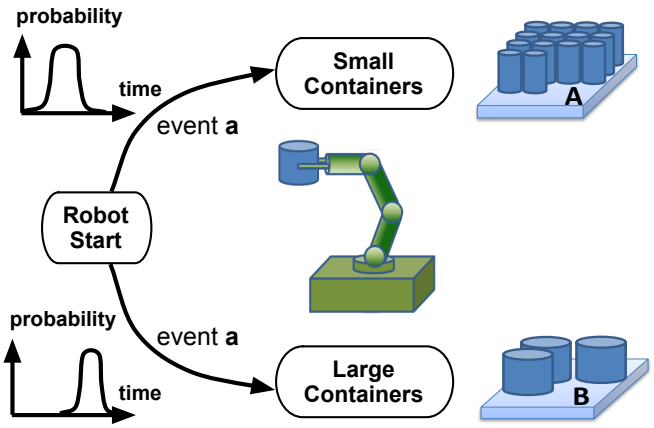


Fig. 2.8 The timing behavior changes based on the container size.

- **Event splitting or timing preprocessing:** Continuing from the previous point, additional question arises that whether the modes are identified during the learning process itself or whether a preprocessing can be used to identify multiple modes and use this information in the learning process, avoiding the additional splitting operation.
- **Timing distribution function:** Finally, an appropriate timing distribution function has to be chosen, which is able to correctly describe the technical processes.

All these issues are analyzed in more detail in the following sections.

Chapter 3

Formalisms

This chapter leads to the topic of this thesis and describes the necessary formalisms. First, we give some basic notions to sketch the background of the work. Then, we introduce several automata formalisms. We begin with the well-known Deterministic Finite Automaton (DFA), then we include probabilistic information that leads to the Probabilistic Deterministic Finite Automaton (PDFA) and finally we will include additional timing information that results in the Probabilistic Deterministic Timed Automaton (PDTA). The PDTA forms the basis for the subsequent chapters.

The chapter is organized as follows. First in Section 3.1, some basic notions are introduced. Then, Section 3.2 gives an overview to some different formalisms for finite automata.

3.1 Basic notions

This thesis is about identifying models for technical (time dependent) systems. Before going more into the detail of modeling formalisms and model identification, the basic notations and definitions are given in this section.

Discrete Event System

A Discrete Event System (DES) is a system that is driven by discrete events. A DES has the following properties:

- The DES can only be in one state at one point in time.
- A discrete event causes the state change of the system.
- It is event-driven. i.e. only discrete events can trigger a state change.

The Discrete Event System is already a special variant of the term *system*. Further variants are *continuous systems*, which only consider a continuous behavior, and *hybrid systems*, which is a combination of discrete and continuous behavior.

Model

According to the definition of [Sta73], a model contains three characteristics:

- **Abstraction:** A model contains only selected features.

- **Homomorphism:** Statements on model elements hold for real world entities.
- **Pragmatics:** A model is always created for a certain purpose.

A model can not replace the original in all its particulars. Therefore, only the needed features have to be modeled.

Example 3. *The modeling of a timed behavior model of a production plant includes events and the corresponding timing constraints. The information about needed material for the construction is not necessary and therefore has not to be included in the model.*

Behavior Model

According to the oxford dictionary a behavior describes *the way in which one acts or conducts oneself, especially towards others* (from http://oxforddictionaries.com). Concerning technical systems, a behavior model describes the reaction of the output variables on changing input variables.

Component

In the rarest cases, the behavior is modeled in one overall stand-alone model. Typically, technical systems (especially huge production plants) consist of several modules. Therefore, a single model is created for each module (component).

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

System Model

A system model holds information about several issues of the system. In addition to a behavior model, the topology of network devices or the mapping of inputs and outputs between components can be stored.

Parallelism Model

From a model learning perspective, a parallelism model divides a system in asynchronously working sub-modules. This is necessary since, in general, it is not possible to learn one overall model for the complete system. Figure 3.1 illustrates the necessity of a parallelism model. In a system with two independent and asynchronously working components, theoretically each state of the first component can occur with each state of the second component. Therefore, the number of states has to be multiplied; in the example: $n = 10 \cdot 5 = 50$ states. Indeed, if a parallelism model is used, each component is identified separately and the number of states has to be summarized; in the example: $n = 10 + 5 = 15$ states.

Simulation

In general, the simulation is an approach for analysis of systems which are too complex for theoretical or formulaic treatment. This is mainly the case in dynamic system behavior. Simulation experiments are carried out on a model to gain insight into the real system.

If finite state machines are used, then a simulation is the computation of a path through the automaton, beginning with the initial state and quitting at a final state.

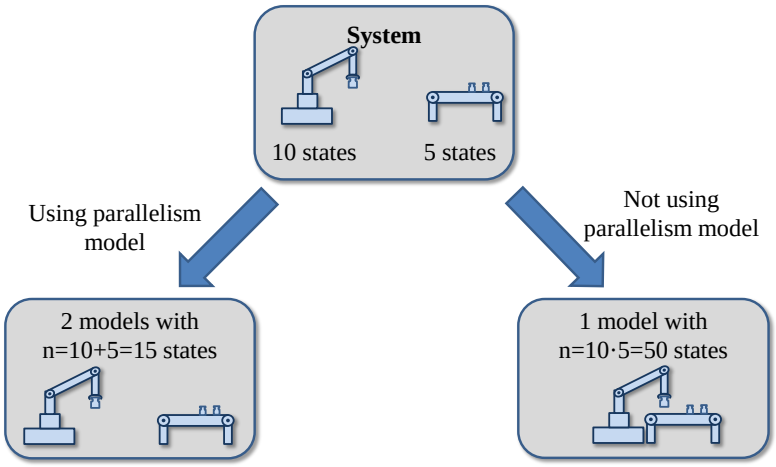


Fig. 3.1 The need of a parallelism model.

Using a simulation of an automaton, new data can be created. This is especially done, if only little data is available to identify a generalized model.

Learning Examples

Definition 2 (Learning Example). A learning example D is a matrix of values:

$$D = \begin{bmatrix} t_1 & q_{1,1} & \dots & q_{d,1} \\ t_2 & q_{1,2} & \dots & q_{d,2} \\ \vdots & \vdots & \vdots & \vdots \\ t_l & q_{1,l} & \dots & q_{d,l} \end{bmatrix},$$

where $t_j, \forall j = 1...l \in \mathbb{N}$ are the time stamps and $q_{i,j}, \forall i = 1...d, j = 1...l \in \mathbb{N}$ are the values for the discrete sensors and actuators in the production plant. $q_{i,j} \in 0, 1$ is the discrete (or binary) value of the i-th signal at the j-th time stamp.

Please note that it is not distinguished between input and output signals.

The learning examples are combined together in a sequence

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

with n recorded examples.

Each learning example includes one event sequence. A single event in this event sequence is defined as follows.

Definition 3 (Event). Let D_i be a learning example according to Definition 2. An event e is defined as a change of one or more discrete signals between two subsequent signal vectors, i.e. $[q_{1,i} \dots q_{d,i}] \neq [q_{1,i+1} \dots q_{d,i+1}]$.

Subsequent events lead to an event sequence:

Definition 4 (Event Sequence). An event sequence E (also referred to as string) is defined as a sequence of n events e_i: $(e_1) \rightarrow (e_2) \rightarrow ... \rightarrow (e_n)$, where e_i is an event according to Definition 3, with $i, n \in \mathbb{N}$. The length of a sequence is defined over the length of the string, i.e. the number n of events e_i.

Language, accepted/ non-accepted words and the Word Problem

A running system creates a sequence of events, also called strings in the field of theoretical computer science. All possible strings, also referred to as accepted words, form a language L. All words which are not contained in the language are called non-accepted or rejected words.

Definition 5 (Language). A language L_A is defined as a set of words which are accepted by an automaton A.

$$L_A = \{\text{w} \mid \text{w is accepted by an automaton } A\}$$

The identification of a language requires the existence of words that are part of the language. These words are provided either as positive or negative learning examples:

Definition 6 (Positive Learning Example). A learning example D is denoted as positive with respect to a language L_A if the probability

$$p(D|L_A) > 0.$$

Positive learning examples are labeled as D_+.

On the contrary, negative learning examples are defined as follows:

Definition 7 (Negative Learning Example). A learning example D is denoted as negative with respect to a language L_A if the probability

$$p(D|L_A) = 0.$$

Negative learning examples are labeled as D_-.

The Word Problem is well known in the field of theoretical computer science:

Definition 8 (Word Problem). Given a language L and a word w (also called string), it is checked whether the word w is part of the language L. Each positive learning example is accepted by the languange, whereas each negative learning example is rejected.

System Identification

The term system identification describes the process of observing a system's behavior and creating a corresponding model automatically. For this, approaches from the field of machine learning, data mining and artificial intelligence are used. In the literature, depending on the field of application, sometimes other terms are used, e.g. language identification, model learning or grammatical inference. The identification process is also referred to as learning process. Optionally, expert knowledge can also be used to refine the model with additional information. This is especially done if this knowledge is available anyway or if the required knowledge is not automatically learnable due to the lack of learning techniques or non-observability. Figure 3.2 shows the general system identification process.

Fault

A fault is a defect within the system. Examples for faults are software bugs, a sensor break down or other random hardware faults.

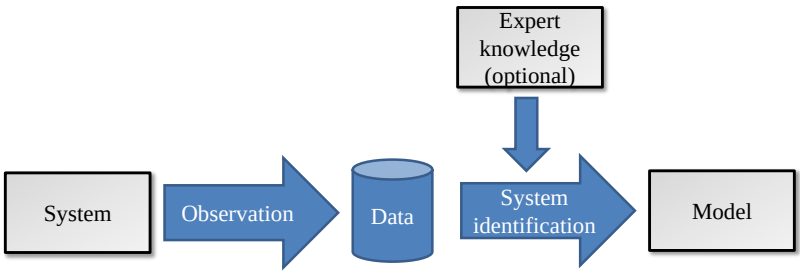


Fig. 3.2 The system identification process.

Error

An error is a deviation of the simulation of a model from the required operation of a system or subsystem. A fault may lead to an error, i.e. an error makes the fault apparent. A fault may also be hidden for some time before it becomes visible as an error.

Failure

A system failure occurs when the system can not perform the required function. The presence of a fault might cause a whole system to deviate from its required operation.

Anomaly

An anomaly is an observed deviation of the model simulation from the real plant behavior. In the literature, the term novelty is also used [MS03a, MS03b], since an anomaly also represents some new behavior, which was not observed before (in the model identification step).

Anomaly Detection

Once the behavior model of the system is learned, it can be used for anomaly detection, i.e. for the identification of unusual behavior. Figure 3.3 shows the typical methodology for anomaly detection: The outputs from the observed system and the predicted output of the model are compared. Differences hint at an anomaly.

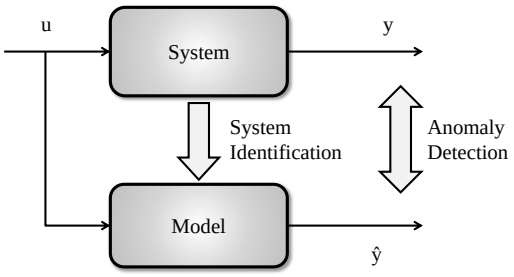


Fig. 3.3 Anomaly Detection in a technical process

Since the anomaly detection will play an important role in this thesis, we define the *anomaly detection problem* formally:

Definition 9 (Anomaly Detection Problem). Given an automaton $\mathcal{A}$ and some observations in form of an event sequence E, according to Definition 4, it has to be decided whether the events e from E belong to the given automaton $\mathcal{A}$. In other words, it is checked whether E is accepted by $\mathcal{A}$.

Note the similarities between Definition 9 and Definition 8, i.e. finding anomalies in the behavior of timed technical systems corresponds to solving the well known Word Problem. This at least holds true if finite automata are used for the model of normal behavior.

Diagnosis, Verification

Model-based diagnosis comprises more actions than the anomaly detection. While anomaly detection contents itself with detecting some anomalous behavior, diagnosis goes a step further and tries to determine the cause for the anomalous behavior. Model-based diagnosis typically comprises three steps [BJ94]:

1. *Symptom Detection:* This step corresponds to the anomaly detection.
2. *Hypothesis Generation:* The listing of potential error causes.
3. *Simulation and Check:* All potential error causes from step 2 are simulated using a modified model. The simulation output is compared with the logs of the anomalous event sequence. If the simulation output corresponds to the anomalous observations, the correct error cause is identified.

In this thesis, however, we confine our self to the detection of anomalies.

3.2 Automata Formalisms

In this work, timed automata are used for the anomaly detection. This Section introduces the formalism step by step, beginning with the basic form of automata (Non-Timed Deterministic Finite Automata) leading to the formalism Probabilistic Deterministic Timed Automaton, which is used in Sections 6 and 7.

3.2.1 (Non-Timed) Deterministic Finite Automata

The **Deterministic Finite Automaton, (DFA)** [TB73] (sometimes referred to as (deterministic) finite state automaton/machine, FSA/FSM) is the basic automaton. All other types of automata, introduced in this section, are based on this formalism.

Definition 10 (Deterministic Finite Automaton). A Deterministic Finite Automaton (DFA) $\mathcal{A}$ is a 5-tuple (S, s_0, F, Σ, T), where

- S is a finite set of states,
- $s_0 \in S$ is the initial state,
- $F \subseteq S$ is a set of final states/ accepted states,
- Σ is a finite set of symbols a and

- T is a set of transitions of $\mathcal{A}$. A transition from state s to s' triggered by $a \in \Sigma$ is represented with (s, a, s').

The DFA basically consists of states and transitions, where an event triggers a transition to change the current state. An event typically is a change of some discrete value in the system (e.g. bottle filled or position reached). An example for a simple DFA can be seen in figure 3.4.

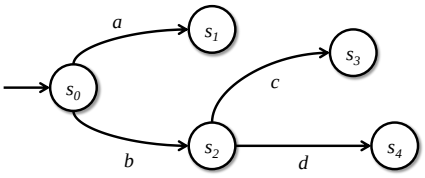


Fig. 3.4 Deterministic Finite Automaton (DFA).

A **Probabilistic Deterministic Finite Automaton** (PDFA) [CO94], also known as Stochastic Finite Automaton (SFA), is a 6-tuple $(S, s_0, F, P, \Sigma, E)$. In addition to the DFA, the PDFA contains a set of probability matrices giving probabilities for the firing of each transition. Probabilistic information is needed if we want to identify an automaton from positive data only.

Definition 11 (Probabilistic Deterministic Finite Automaton). A Probabilistic Deterministic Finite Automaton (PDFA) $\mathcal{A}$ is a 5-tuple $(S, s_0, F, P, \Sigma, T)$, where

- S is a finite set of states, each state s includes a probability of leaving the automaton. If a state s_i is not a final state, the probability $p(i) = 0$, otherwise the probability is from the interval $(0..1]$,
- $s_0 \in S$ is the initial state,
- $F \subseteq S$ is a set of final states/ accepted states,
- Σ is a finite set of symbols a,
- T is a set of transitions of $\mathcal{A}$. A transition from state s to s' triggered by $a \in \Sigma$ is represented with (s, a, s') and
- P is a set of probability matrices, $p_{i,j}(a)$ gives the probability of taking the transition from state s_i to state s_j with an input symbol a. With $p(i)$ for the probability that a string ends at the state s_i it holds that:

$$p(i) + \sum_{\forall s_i, s_j \in S, \forall a \in \Sigma} p_{i,j}(a) = 1 \tag{3.1}$$

3.2.2 Nondeterministic Finite Automata

Apart from Deterministic Finite Automata, there also exist Nondeterministic Finite Automata (NFA). Unlike in deterministic automata, the number of outgoing transitions in one state is not restricted to only one for each symbol. Each state can have

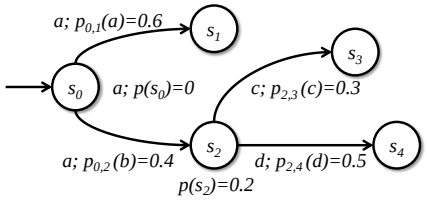


Fig. 3.5 Probabilistic Deterministic Automaton (PDFA)

multiple outgoing transitions with the same symbol. In addition to that, NFAs contain not only a single initial state $s_0 \in S$ but a set of initial states $S_0 \subseteq S$.

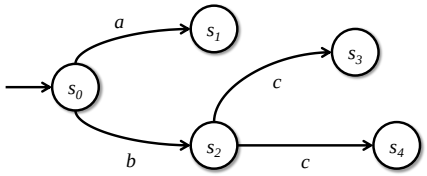


Fig. 3.6 Nondeterministic Finite Automaton (NFA)

Since NFAs are not in the scope of this thesis, we do not give a formal definition.

3.2.3 Timed Automata

Naturally, technical systems depend on time. To model this kind of systems, a more powerful formalism is needed that also includes timing information.

We can define a timed sequence according to the event sequences from Definition 4 . Since we always acquire the data with finite time precision, the sequence of timed events (Timed Sequence) can now be described as value-timestamp-pairs:

Definition 12 (Timed Sequence). Let e_i be an event according to Definition 3 and t_i the corresponding (relative) timestamp. A timed sequence (also timed string) τ consists of n pairs of an event e_i with a timestamp t_i: $(e_i, t_i) \in \Sigma \times N$: $(e_1, t_1) \rightarrow (e_2, t_2) \rightarrow ... \rightarrow (e_i, t_i) \rightarrow ... \rightarrow (e_n, t_n)$. The length of a timed sequence is not defined over the passed time but over the length n of the string, which is the number of events.

The **Timed Automaton** (TA) was firstly introduced by Alur and Dill in [AD94]. Additional to the events, the transitions obtain a time guard that has to be fulfilled and which is reset each time the transition is fired. It also allows the existence of multiple clocks and a different set of clocks can be reset in each transition.

Definition 13 (n-clock Deterministic Timed Automaton). An n-clock Deterministic Timed Automaton (n-TA) is a 6-tuple $(S, s_0, F, \Sigma, C, T)$, where

- S is a finite set of states. If a state is not a final state, the probability $p = 0$, otherwise the probability is from the interval $(0..1]$,
- $s_0 \in S$ is the initial state,
- $F \subseteq S$ is a set of final states/ accepted states,
- Σ is a finite set of symbols a,
- C is a set of clocks,
- T is a set of transitions of $\mathcal{A}$. A transition from state s to s' triggered by $a \in \Sigma$ is represented with $(s, a, \lambda, \delta, s')$, where λ specifies the clocks to be reset after this transition and δ is the clock constraint over C. The automaton changes from state s_i to state s_j triggered by $a \in \Sigma$ if the current clock value satisfies δ. The clocks of λ are set to 0 after executing a transition, so that these clocks start counting time from executing this transition.

Remark 1. The representation of the timing information is not specified in the definition. Usually, time ranges are used, i.e. the minimum and maximum time stamp is stored for each event. The values in between are distributed uniformly. The determination of the minimum and maximum time value can be done globally for each event in the automaton or locally for each transition. However, the timing behavior in technical systems (and in most other real-world examples, too) shows a different behavior. The timing behavior mostly shows a behavior following a Gaussian distribution. In this case, the mean value and the standard deviation for the timing have to be calculated in each transition.

Based on this formalism of the Timed Automaton, there exist some modified versions. Here, we only give those, which are used in this thesis.

One-Clock Timed Automaton

Technical systems mostly depend on one clock only, the global time base. Additionally, the identification of multiple clocks poses some additional problems when it comes to the identification of Timed Automata. Verwer already showed that in contrast to n-clock Timed Automata, Timed Automata with only one clock can be identified efficiently [Ver10]. Therefore, the formalism of One-Clock Timed Automaton was introduced.

Definition 14 (One-Clock Deterministic Timed Automaton). A One-Clock Deterministic Timed Automaton (1-DTA) is a Timed Automaton according to Definition 13, where $|C|= 1$, i.e. only one clock is used. In each transition $\lambda = c$, i.e. the clock is reset after each firing of a transition.

The reset of the clock after each firing of a transition leads to a relative time base.

Probabilistic Deterministic Timed Automaton

The introduction of a Probabilistic Deterministic Timed Automaton (PDTA) is based on the same motivation as for the untimed automaton: we want to identify an automaton from positive data only.

The PDTA is a One-Clock Timed Automaton with probabilistic information. It allows only one clock, which is reset at each transition firing.

Definition 15 (Probabilistic Deterministic Timed Automaton). A Probabilistic Deterministic Timed Automaton (PDTA) is a 6-tuple $(S, s_0, F, P, \Sigma, T)$, where

- S is a finite set of states, where each state s includes a probability of leaving the automaton. If a state is not a final state, the probability $p = 0$, otherwise the probability is from the interval $(0..1]$,
- $s_0 \in S$ is the initial state,
- $F \subseteq S$ is a set of final states/ accepted states,
- Σ is a finite set of symbols a,
- T is a set of transitions. A transition is represented with (s, a, δ, s'), where δ is the clock constraint. The automaton changes from state s_i to state s_j triggered by a symbol $a \in \Sigma$ if the current clock value satisfies δ. After executing a transition, the clock c is set to 0, so that the clock starts counting time from executing this transition.
- P is a set of probability matrices, $p_{i,j}(a)$ gives the probability of taking the transition from state s_i to state s_j triggered by $a \in \Sigma$. With the probability $p(i)$ that a string ends at the state s_i it holds that:

$$p(i) + \sum_{\forall s_i, s_j \in S, \forall a \in \Sigma} p_{i,j}(a) = 1 \tag{3.2}$$

An example PDTA is depicted in Figure 3.7. The clock guard is given here as time range of possible relative time values with the minimum and maximum time stamp.

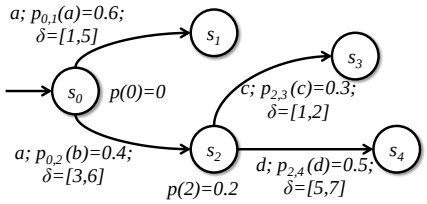


Fig. 3.7 Probabilistic Deterministic Timed Automaton (PDTA)

There exist some further subclasses of Timed Automata. Since these are not used in this thesis, only a rough overview is given without formal definition. A detailed overview to these formalisms can be found in [CL06].

Event-Recording Automata (ERA) as a subclass of TAs were introduced by Alur et. al. in 1999 [AFH99]. In an ERA, every event a contains a clock that records the time since the last occurrence of a.

Edge Labeled Continuous-time Markov Chains ($CTMC_L$) are a mixture of TAs and Hidden Marcov Models (HMM). Strictly speaking these are not FSMs since this formalism is based on Markov Chains, but it is very similar to timed FSM; transitions are labeled and time information is included. In each state, two probabilities are given: *(1)* the probability of which state to choose next and *(2)* the probability of how long to remain in the current state.

3.2.4 Probabilistic Deterministic Hybrid Automaton

In contrast to the basic automata, the **Probabilistic Deterministic Hybrid Timed Automaton** (PHyTA, basically introduced by Alur in 1995 [ACH+95]) considers not only signals with discrete values but also continuous value changes. To describe the continuous behavior, differential equations can be used. As long as a system remains in a certain state, the behavior is described by a set of differential equations. Usually hybrid automata are restricted to linear dependencies, but more complex functions are also possible.

Definition 16 (Probabilistic Deterministic Hybrid Timed Automaton). A Probabilistic Deterministic Hybrid Timed Automaton (PHyTA) is a tuple $A = (S, s_0, F, \Sigma, T, \Delta, c, \Theta)$, 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 the alphabet comprising all relevant events.
- $T \subseteq S \times \Sigma \times \Delta \times S$ gives the set of transitions, where Δ is the set of timing constraints δ_i. E.g. for a transition $\langle s, a, \delta s' \rangle$, $s, s' \in S$ are the source and destination states, $a \in \Sigma$ is the trigger event and δ is the timing constraint.
- A single clock c is used to record the time evolution. The clock is reset at each transition. Therefore, only relative time steps are possible.
- A set of functions Θ with elements $\theta_s : \mathbf{R}^n \rightarrow \mathbf{R}^m, \forall s \in S, n, m \in \mathbf{N}$, i.e. θ_s is the function that computes signal value changes within a single state s.

Figure 3.8 shows a small example of a Probabilistic Deterministic Hybrid Timed Automaton. The two states are labeled with "conveyer belt is running" and "conveyer belt is halted". As a condition for the transition from the first to the second state, the conveyer belt has to be stopped within 50 time units (for instance: seconds) measured from the entrance in the first state (relative time). Additionally a probability of $p = 0.8$ is given that this event will be carried out. Inside the states, continuous signals can change over time, e.g. while the conveyer belt is running (in state 1), a container is filled with some material and the weight is rising, which is represented by the rising curve. If a continuous signal does not change its value, the function value is constant over time.

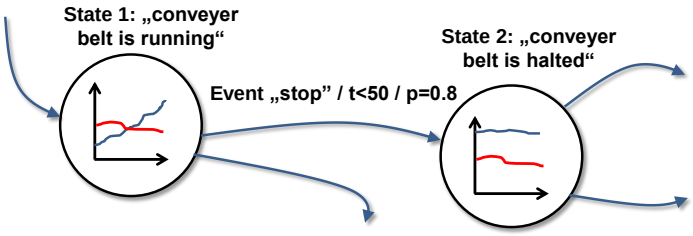


Fig. 3.8 An example of a Probabilistic Deterministic Hybrid Timed Automaton.

Chapter 4

Complexity of Identification of Finite Automata

The identification of (deterministic) finite automata is an active research field and some work has already been done to analyze the complexity of these identification processes.

In this chapter, we summarize the most important issues from the current state of the art concerning the complexity of identification of finite automata. For this, different identification frameworks have been introduced, which help to classify the identification algorithms. These identification frameworks allow to give generalized statements about the complexity. For the context of this thesis, the framework of *identification in the limit* is an important issue. This is used to evaluate the complexity of identification of untimed and timed automata.

In this chapter, the main definitions and known results for the complexity of automaton identification are the following:

- Definition of the automaton identification problem
- Complexity of the automaton identification problem, which is known to be NP-complete
- Definition of the identification frameworks
- Complexity of identification of timed automata
- Different results for 1-clock timed automata and n-clock timed automata

This chapter is organized as follows: First in Section 4.1, we give a definition of the 'identification problem', which follows from the representation of information to the *automaton identification problem*. In Section 4.2, we give an overview to the identification frameworks, which are used to group identification algorithms according to specific features. Finally in Section 4.3, we summarize some work concerning the identification complexity. For this, we start with the complexity of identification of (untimed) DFA and continue to the complexity of identification of Timed Automata.

4.1 The Identification Problem

Before considering the identification problem itself, this section starts with the representation of information. Naturally, the information is represented in form of strings (a string is a sequence of characters) in the field of grammatical inference. However, there are different possibilities to express some kind of information in some cases. For instance, the string "abcdefghijklmnopqrstuvwxyz" can be easily expressed as "the lower-case characters from the alphabet".

The Kolmogorov complexity is a measure of the information content of a string and is given by the length of the shortest program that generates this string. This shortest program is thus the best compression of the string without loss of information. If the Kolmogorov complexity of a character string is at least as large as the string itself, the string is then referred to as incompressible, random or unstructured. The closer the Kolmogorov complexity is to the length of the string, the more 'random' is the string (and the more information it contains). The string "abcabcabcabcabcabcabcabcabc", for instance, can be expressed as "10 times abc", which now uses 12 (including white spaces) characters instead of 30. However, there also exist examples of strings that cannot be expressed in a simpler way than just repeating their characters. The principle of Kolmogorov complexity was independently developed in 1963 by Andrei Kolmogorov [Kol63, Kol98], in 1964 by RJ Solomonoff [Sol64], and has connections to Shannon's information theory [Sha48], which uses the concept of entropy to characterize the density of information (information content).

The procedure of transferring the original string to the dense form can be denoted as '*learning*'. Learning in this context means to represent given information in a compact form. From the Kolmogorov complexity point of view, learning means to find rules that describe the generation of strings. This works best for repeating sub-strings, e.g. the sub-string "abc" in the first example, but also for well known character sequences as the alphabet "abc...xyz". It even becomes more important when many combinations of characters (or character sequences) are allowed to form an accepted word.

It can be seen that the listing of all possible information strings is not a good solution. The following example uses an artificial problem to illustrate the application scenario.

Example 4. *Given some input strings with repeating character sequences, the task is to reduce these to a representation with the smallest Kolmogorov complexity. Four example input strings are listed in Figure 4.1 in the first line. One may notice that it is natural to use regular expressions for this purpose. The second line gives the resulting regular expression. Under the assumption that the repeating character sequence can occur arbitrarily often, the regular expression can be simplified a bit more. Each regular expression can be visualized as an automaton. The regular expression from this example is visualized in the third line in Figure 4.1.*

Example 4 illustrates that *learning* in the context of this thesis can be considered as: reduction of the size of the input data strings to a representation with the smallest Kolmogorov complexity, which is considered to be a regular grammar, visualized as finite automaton.

In literature, the terms *learning* and *identification* are used synonymously. In most cases *learning* denotes the gain of knowledge, while *identification* means the finding of an unknown structure, e.g. the automaton structure with states and transitions. In this thesis, these both terms are also used.

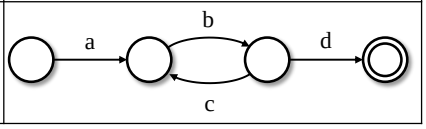
Input strings	1. abcbcbcbcbcbd 2. abd 3. abcbcbcbd 4. abcbcbcbcbcbcbcbcbcbcbcbcbcbd
Regular expression	a(bc)$^{\{0,13\}}$bd more generalized: a(bc)*bd
Automaton	

Fig. 4.1 Analogy between the given input strings and the Kolmogorov complexity as regular expression.

The methods in automaton identification are intended to learn the structure of the model. Other approaches assume a given structure and only identify the parameters (for instance probabilities in Bayesian networks or weight values in neural networks). The automaton identification problem is the task to identify the unknown structure, which corresponds to find an automaton that is consistent with some input data. According to Occam's razor [Dom99], this should be represented in the smallest possible form, i.e. as few states as possible. A smaller automaton is simpler to understand and therefore it is favored to represent the observed behavior.

Definition 17 (Automaton Identification Problem P_{AI}). Given finite disjunctive sets of positive D_+ and negative D_- examples (according to Definition 6 and Definition 7 respectively) and a positive integer k, then the automaton identification problem P_{AI} is the problem of identifying an automaton A with at most k states (called the smallest automaton), where the identified automaton A is consistent with the sets of data examples, i.e. A accepts all examples from D_+ and rejects all examples from D_-.

Gold showed in [Gol78] that the automaton identification problem P_{AI} for DFAs is NP-complete.

Thus, under the assumption that $P \neq NP$, no algorithm exists that identifies a DFA efficiently from given data. Hence, the problem of finding the smallest possible DFA without knowing the number of states, which is an optimization variant of the automaton identification problem with a fixed number of states, cannot be easier to solve.

On top of this result, in [PW93] Pitt and Warmuth showed that *the minimum consistent DFA problem cannot even be approximated within any polynomial*.

Definition 18 (The Problem of identifying an Approximately Small Automaton $P_{AI_{approx}}$). Is there an algorithm, that can identify (or just determine the existence of) a consistent DFA of size $p(|S_{min}|)$ for some polynomial function p, where $|S_{min}|$ is the size of the smallest DFA, which is consistent with the given finite set of positive and negative examples $(\mathcal{D}_+, \mathcal{D}_-)$?

The problem of identifying approximately small DFA $P_{AI_{approx}}$ has been shown to be NP-complete [PW93] .

In [Vod13] it is shown that the aforementioned results can be generalized to the class of TAs. Since a DFA corresponds to a TA without timing constraints, the set of clocks is set to $X = \emptyset$ and the timing constraints $\Delta = \emptyset$. A TA without timing constraints behaves like a DFA and the given results apply for TA.

4.2 Identification frameworks

The results from the last section suggest that automata can not be identified automatically from given data in an efficient way. Then the question arises whether finite automata can be still identified efficiently, if more examples can be provided. Therefore, some identification frameworks have been introduced that offer more promising results. Here, the frameworks are introduced informally. In later sections, formal definitions will be given for the used frameworks.

Identification from given finite data

The framework of *identification from given finite data* is used when the amount of data at disposal is limited, i.e. it is not possible to acquire additional data even if it is needed. The goal is to identify the target model as accurate as possible. The target model should represent the data and classify them correctly. The framework of *identification from given finite data* was introduced in [Gol78] to determine the complexity of automaton identification.

Identification in the limit

In contrast to the framework of 'identification from given finite data', the framework *identification in the limit* assumes that an access to a data source is available such that additional data can be requested as required. The learner is getting more and more samples to learn the target language. Each sample improves the quality of the target model. At some point, after a finite number of observations (in the limit) the model reaches the target language. Even if the data source provides an infinite amount of data, the identification process converges in finite time. This identification is called *identification in the limit* [Gol67].

Figure 4.2 illustrates the difference between the frameworks *(a) identification from given finite data* and *(b) identification in the limit*.

Query learning

Both identification frameworks, 'identification from given finite data' and 'identification in the limit', work as passive learner (will be introduced in Section 5.2.1). The data are taken as they are provided (e.g. as stored in the data base or as taken from the data stream during data acquisition). In contrast, the framework of *query learning* works in an active manner. That means, data can be requested on demand. This implies a student-teacher relationship. The student (learner) can make a query to the teacher (Oracle). This query can either be a question whether a certain sample is accepted by the target language (answer: yes or no) or whether the hypothesis is consistent with the target model (answer: yes or counterexample). This framework assumes the existence of an Oracle. The Oracle knows the target language and answers the questions correctly. Figure 4.3 illustrates the difference between active and passive learning according to [Ton01].

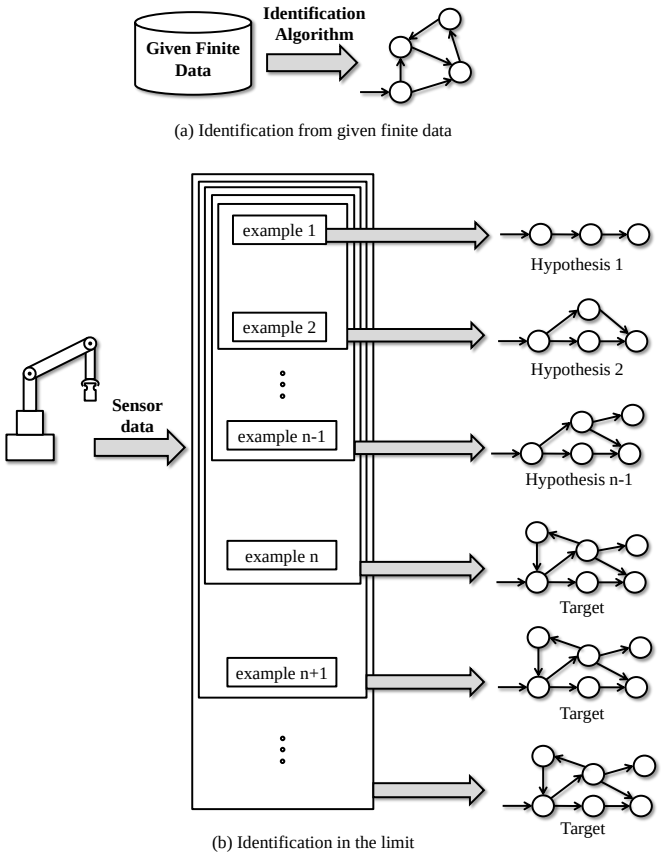


Fig. 4.2 Identification from given data and identification in the limit [Vod13].

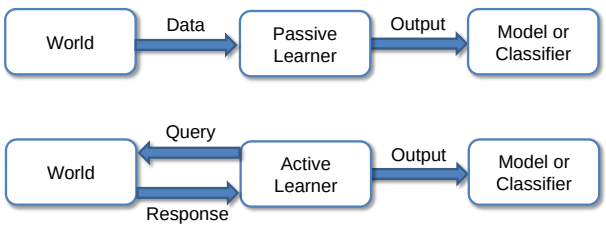


Fig. 4.3 Difference between active and passive learning [Ton01].

PAC identification

All three aforementioned identification frameworks are intended to identify a correct automaton from data. The framework of *Probably Approximately Correct learning* (PAC-learning, introduced by Leslie Valiant in 1984 [Val84]) allows an acceptable deviation. The deviation of the identified model from the target model is only allowed with a given probability. Even a deviation larger than the predefined value is possible, but only with a very small probability. PAC-learning can be used instead of the aforementioned identification frameworks if *(1)* the target language can not be identified correctly, *(2)* the needed amount of data for convergence is overwhelming, or *(3)* when the runtime of the present identification algorithm is inefficient.

Algorithms that use the framework of PAC-learning are given e.g. in [CG08] and [CT04].

The focus in this thesis is more on the framework of *identification in the limit*. The other frameworks are considered marginally.

4.3 Complexity of Identification of Finite Automata

Section 4.1 showed that DFAs can not be identified efficiently. In this section, the complexity of identification of finite automata is analyzed in more detail. For this, especially the framework of identification in the limit is used.

4.3.1 Identification in the Limit

In Section 4.2, the framework of identification in the limit was introduced informally. Here, we give a formal definition and refine this to stronger limitations.

Definition 19 (Identification in the limit [Gol67]). In the framework of identification in the limit, the learner gets more and more data. If there exists an algorithm A and during the identification process the model converges to the target language L_t, the algorithm A identifies L_t in the limit.

However, the framework of identification in the limit itself gives no information about the time or amount of data needed for the identification process. Therefore, two more restricted versions are introduced.

Definition 20 (Weak Polynomial Identification in the Limit with Probability One [dlH10]). Let D be a set of observations/ learning examples according to Definition 2 for which an automaton $\mathcal{A}$ should be identified. An automaton learning algorithm A identifies $\mathcal{A}$ weakly polynomially in the limit with probability one

1. if A identifies $\mathcal{A}$ in the limit with probability one and
2. if the identification runtime is in $O(f(|D|))$,

where $|D|$ denotes the length of all observations and f a polynomial function.

Simplified, it can be said: A language is called *weakly polynomially identifiable in the limit with probability one* if the language is identifiable in the limit with time polynomial to the size of the input data.

The definition of weak polynomial learning in the limit gives information about the time needed for learning, but still holds no information about the needed amount of data. This leads to the more restricted version of *strong polynomial identification in the limit*.

Definition 21 (Strong Polynomial Identification in the Limit with Probability One [dlH10]). Let D be a set of observations/ learning examples according to Definition 2 for which an automaton $\mathcal{A}$ should be identified. An automaton identification algorithm A identifies $\mathcal{A}$ strongly polynomially in the limit with probability one

1. if A identifies $\mathcal{A}$ in the limit with probability one and
2. if the identification runtime is in $O(f_1(|D|))$ and
3. if $|D| \leq f_2(|\mathcal{A}|)$,

where $|D|$ denotes the length of all observations, $|\mathcal{A}|$ denotes the size of the final automaton, i.e. the number of states and f_1, f_2 are polynomial functions.

Simplified, it can be said: A language L is called *strongly polynomially identifiable in the limit with probability one* if the language is identifiable in the limit with time polynomial to the size of the input data and using only a polynomial number of input samples (polynomial to the size of the final automaton). This is also referred to as *identifiable from polynomial time and data* or simply *efficiently identifiable in the limit*. It is already proved that DFAs are identifiable in the limit.

Theorem 1. *DFAs are weakly identifiable in the limit using positive and negative learning examples [Gol67, Pit89].*

Proof (Sketch). To prove this theorem, Gold uses the identification by enumeration technique. Step-by-step, the examples are included to the automaton and the algorithm always delivers the smallest automaton. The automaton is consistent with all data examples, which so far have been used for the identification. The technique is computationally inefficient. However, it identifies DFAs in the limit. □

In general, this seems to be a good result. However, the algorithm uses positive as well as negative learning examples. Since our goal is to identify behavior models in cyber-physical production systems, we can not use negative learning examples and therefore we are more interested in the identification using positive learning examples only. Unfortunately, the results given in the next theorem, are less promising.

Theorem 2. *DFAs are not identifiable in the limit using positive learning examples only [Gol67, Pit89].*

Proof (Sketch). Generally, DFAs belong to a class of languages that contain all finite and at least one infinite language. To prove the theorem, Gold constructs a text for learning such an infinite language in the limit that constantly repeats a text for learning some of the finite languages from the same class. Since the language is infinite, the repetitions can occur an infinite number of times and therefore the identification algorithm mistakenly outputs the finite language indefinitely long. From this, it follows that DFAs are not identifiable in the limit using positive learning examples only. □

According to [Vod13], Theorem 1 and Theorem 2 can also be generalized to the class of timed automata. Since a DFA corresponds to a TA without timing constraints, the set of clocks is set to $X = \emptyset$ and the timing constraints $\Delta = \emptyset$. A TA without timing constraints behaves like a DFA and the results of Theorem 1 and Theorem 2 apply for TA.

In [OG92], it is further shown that the class of all DFAs are efficiently identifiable in the limit using a state merging method [OG92]. Further, in [DLH97] it is proven that NFAs are not efficiently identifiable in the limit. The proofs can be found in the cited literature and are not given here.

4.3.2 Efficient Identification of Timed Automata

Much work has been done on complexity of identification of DFA. On the contrary, less work has been done on the complexity of identification of TA. In his dissertation, Verwer has dealt thoroughly with this subject. This section summarizes the main contributions to the complexity of the identification of timed automata according to [VWW08], [Ver10].

Section 4.3.1 introduced the framework of identification in the limit. In [VWW08], Verwer gives alternative definition for *efficient identification in the limit* using characteristic sets.

Definition 22 (Characteristic Set). A characteristic set $\mathcal{D}_c$ of a target language L_t for an algorithm A is a finite set of examples $(\mathcal{D}_{c+} \in L_t, \mathcal{D}_{c-} \in \overline{L_t})$ such that:

- given the characteristic set $\mathcal{D}_c$ as input, the algorithm A identifies the target language L_t correctly, i.e the algorithm A returns an automaton A_1, such that $L(A_1) = L_t$.
- given any other input sample $\mathcal{D}' \supseteq \mathcal{D}_c$, L_t is still identified correctly.

Verwer uses the characteristic sets to prove the efficient identifiability of a certain class of automata. A class of automata C is *efficiently identifiable in the limit* if there exists an algorithm A and two polynomials p and q, such that for any input sample of size n, the runtime of A is bounded by $p(n)$ and for every target language $L_t = L(\mathcal{A}), \mathcal{A} \in C$ there exists a characteristic set $\mathcal{D}_c$ of L_t, the size of $\mathcal{D}_c$ is bounded by $q(|\mathcal{A}|)$.

Definition 23 (Polynomial reachability). A class of automata C is called polynomially reachable if for all automata $\mathcal{A} \in C$ for any reachable state q there exists a string τ and a polynomial function p, with $|\tau| \leq p(|\mathcal{A}|)$, such that τ reaches q in $\mathcal{A}$. (For timed strings recall Definition 12.)

Proposition 1. *The class of DTAs is not polynomially reachable [VWW08].*

Proof. This can be proven by giving an example automaton, which is not polynomially reachable. In [VWW08], Verwer gives the automaton as illustrated in Figure 4.4. In order to reach state q_3, a string of length $|\tau| \geq 2^n$ is required, since it can only be reached if both clock guards $x \geq 2^n$ and $y \leq 1$ are satisfied. For this, the clock guard y has to be reset 2^n times. Thus, the shortest string reaching state q_3 is of exponential size. From this, it follows that the class of timed automata is not polynomially reachable. □

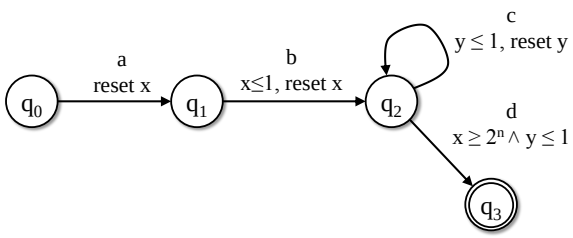


Fig. 4.4 Example automaton to demonstrate that timed automata are not polynomially reachable (according to [VWW08]).

The polynomial reachability is required for the polynomial distinguishability.

Definition 24 (Polynomial distinguishability). A class of automata C is called polynomially distinguishable if for any two automata $\mathcal{A}_1, \mathcal{A}_2 \in C$ and $L(\mathcal{A}_1) \neq L(\mathcal{A}_2)$ there exists a polynomial function p and a string $\tau \in L(\mathcal{A}_1)\Delta L(\mathcal{A}_2)$, such that $|\tau| \leq p(|\mathcal{A}_1|+|\mathcal{A}_2|)$, where Δ is an operator to construct the symmetric difference of two strings.

Proposition 2. *The class of DTAs is not polynomially distinguishable [VWW08].*

Proof. In Proposition 1, it was shown that the class of DTAs is not polynomially reachable. Thus, there is a DTA $\mathcal{A}$ with a state q for which the length of the string τ can not be bounded by a polynomial $p(|\mathcal{A}|)$. Taking this DTA $\mathcal{A}$ and constructing two DTAs $\mathcal{A}_1 = \langle S, X, \Sigma, \Delta, q_0\{q\}\rangle$ and $\mathcal{A}_2 = \langle S, X, \Sigma, \Delta, q_0\{\emptyset\}\rangle$ (by construction, τ is the shortest string in $L(\mathcal{A}_\infty)$ and $L(\mathcal{A}_\in)$ accepts the empty language), τ is the shortest string such that $\tau \in L(\mathcal{A}_1)\Delta L(\mathcal{A}_\in)$. Since $|\mathcal{A}_1|+|\mathcal{A}_2| \leq 2 \cdot |\mathcal{A}|$, the length of τ can not be bounded by a polynomial $p(|\mathcal{A}_1|+|\mathcal{A}_2|)$, and therefore, the class of DTAs is not polynomially distinguishable. □

The terms of polynomial reachability and distinguishability are now used to show that, in general, the class of timed automata can not be identified efficiently.

Theorem 3. *The class of DTAs cannot be identified efficiently [Ver10]*

Proof. Two conditions have to be fulfilled to satisfy the claim of efficient identification in the limit: The class of automata has to be (1) polynomially reachable and (2) polynomially distinguishable, since to be able to efficiently identify a DTA $\mathcal{A}$, we have to be able to distinguish $\mathcal{A}$ from any other DTA $\mathcal{A}_1$ using a (timed) string that is bounded by a polynomial p.

In Proposition 1, it has been shown that the class of DTAs is not polynomially reachable, further in Proposition 2, it was shown that the class of DTAs is not polynomially distinguishable. Therefore, the class of DTAs cannot be identified efficiently.

□

Theorem 3 at least applies for timed automata with more than one clock, since Proposition 1 requires an automaton with more than one clock.

In his dissertation, Verwer gives an algorithm that learns an n-clock DTA. It is based on the algorithm ID_1-DTA (see Section 5.2.4), which identifies a 1-DTA.

Verwer shows that the modification of the algorithm ID_1-DTA identifies an n-DTA in time polynomial to the size of the input and exponential to the amount of clocks [Ver10].

Theorem 3 showed, that n-clock DTAs are not identifiable efficiently. Using timed automata with only one clock according to Definition 14, we obtain better results. Verwer proved that the class of one-clock timed automata (1-DTA) is identifiable efficiently [VdWW09]. The proof follows the same strategy as the proof that n-clock timed automata are not efficiently identifiable. First, he showed that 1-DTAs are polynomially reachable, than he proved that 1-DTAs are polynomially distinguishable. This leads to the final theorem, that 1-DTA are efficiently identifiable. Since the proofs comprise several pages in [Ver10], only proof sketches are given here. The full proofs can be found in the mentioned literature.

Proposition 3. *1-DTAs are polynomially reachable [VdWW09]*

Proof (Sketch). Verwer proves it by analyzing for each state s_i in an automaton $\mathcal{A}$, how it can be reached when shortest timed strings τ_i are given . For each state $s \in S$, the number of prefixes of τ that end in s maximally equals the number of times the clock is reset by τ. Furthermore, he shows that the clock is reset by τ at most $|S|$ times. Together it means that τ visits each state at most $|S|$ times during a computation of $\mathcal{A}$. Therefore, the length of τ is polynomially bounded by $|S|\cdot|S|$. □

Proposition 4. *1-DTAs are polynomially distinguishable [VdWW09].*

Proof (Sketch). The proof uses a similar argument as the proof of Proposition 3, but turns at to be more difficult. Considering the difference of two one-clock timed automata, now two clocks have to be checked. However, there is no clock guard that is bounded by both clocks, and automata such as displayed in Figure 4.4 can not be constructed. □

Theorem 4. *DTAs with a single clock are efficiently identifiable [Ver10]*

Proof (Sketch).

This follows from Proposition 3 and Proposition 4. Further in [VdWW09], Verwer gives the algorithm ID_1-DTA (for a short description see also Section 5.2.4), which identifies a 1-DTA and satisfies the following properties:

- *the identification of a single transition needs time polynomial to the size of the input samples,*
- *the amount of needed transition identifications (input data) is also polynomial to the size of the input samples,*
- *for each transition in the automaton, there exists a characteristic D_c set in the size of the smallest 1-DTA, where D_c is consistent with the input data and guarantees that the target language is identified correctly,*
- *the number of these transition identifications that are necessary to identify the correct target language is polynomial to the size of the smallest 1-DTA.*

Combining this with Definition 21, ID_1-DTA identifies a 1-DTA in polynomial time with polynomial data and is therefore efficiently identifying a 1-DTA. More generally, 1-DTAs are efficiently identifiable.

□

Obtaining this positive result, the question arises, whether an n-DTA can also be identified efficiently and whether an n-DTA can be converted into a 1-DTA. The (maybe surprising) answer is that 1-DTAs and n-DTAs are language equivalent [Ver10], however the conversion itself can not be performed efficiently since the 1-DTA is, in worst case, exponentially larger than the n-DTA.

Definition 25 (Language equivalence). Two automata classes C_1 and C_2 are called language equivalent if for each $A_1 \in C_1$ there exists an $A_2 \in C_2$ such that $L(A_1) = L(A_2)$ and vice versa.

Theorem 5. *1-DTAs and n-DTAs are language equivalent [Ver10].*

Proof (Sketch).

To prove this theorem, Verwer describes a method to transform any n-clock automaton to a one-clock automaton. For this, he used a modified region construction method, based on the first version of Alur [AD94], applying it to all but one clocks of the n-DTA. The removed clocks in each state are represented by a constant deviation from the remaining clock. This method does not influence the accepted language. From this it follows that 1-DTAs and n-DTAs are language equivalent.

□

Summarized, it can be said that it is not necessary to deal with the identification of n-DTAs since 1-DTAs and n-DTAs are language equivalent (Theorem 5), while n-DTAs can not be identified efficiently (Theorem 3) and 1-DTAs can be identified efficiently (Theorem 4). Therefore in the further course of this thesis, we only consider the identification of 1-DTAs.

Chapter 5

Identification of Automata and Model-Based Anomaly Detection

This chapter gives an overview of the state of the art for automaton identification algorithms and their application, the anomaly detection in cyber-physical production systems.

The automaton identification algorithms are classified according to different characteristics. The classification and an overview to the most important algorithms is given in this section. The identification algorithms BUTLA and OTALA, which are introduced in this thesis (see Chapter 6), are partially based on these algorithms. Due to the designated application, a special focus is on the applicability of the algorithms to anomaly detection in cyber-physical production systems.

The main contents of this chapter are the following:

- We give an extended classification scheme for finite automaton identification algorithms.
- Based on the classification, existing learning algorithms are explained.
- We identify the gaps, which are addressed later in this thesis.

This chapter is organized as follows:

Section 5.1 addresses the actual application of the identified automata and gives an overview to the state of the art for model-based anomaly detection.

In Section 5.2, a classification of learning algorithms is given and based on this classification some state of the art identification algorithms are presented. Finally in Section 5.3, we conclude this part and point out the gaps in the presented automaton identification algorithms, which are addressed in this thesis.

5.1 Approaches for Anomaly Detection

Anomaly detection approaches can be subdivided into two groups: *(1)* Phenomenological and *(2)* Model-based approaches.

Phenomenological Approach:
The system's output is directly classified as correct or anomalous. In such approaches, the classifier (e.g. k-nearest neighbor) is trained with learning patterns. In the following classification phase, anomalies are detected by calculating the membership to one of the learned classes (good or anomalous) i.e. anomalies are detected (see figure 5.1 right hand side).

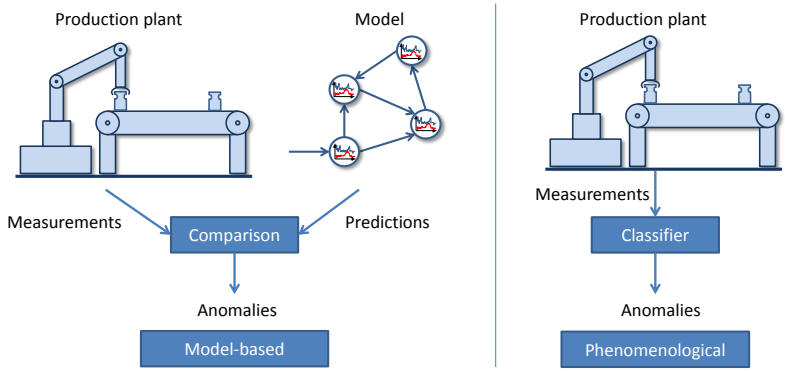


Fig. 5.1 Model-based and Phenomenological Approach.

Model-based Approach:
In order to detect anomalies in the behavior, a model-based approach, as depicted in Figure 5.1 (left hand side), can be used. A model is used to predict the normal behavior of a plant. For this, the simulation model needs all the inputs of the plant, e.g. product information, plant configuration, plant status and sensor/ actuator values. If the actual behavior varies significantly from the simulation results, the behavior is classified as anomalous.

While phenomenological approaches are often more straight-forward and do not require a model, they have one major inherent drawback: They must deduce against the direction of causality since they deduce from measurements (i.e. symptoms) to anomalies. For complex distributed systems with many inter-dependencies between components and complex causalities, this leads to several problems: *(1)* The classification rules need a high number of measurement variables – including the measurements' history – to differentiate between error causes. *(2)* A high number of classification rules is needed to capture the effect of different input combinations and again all combinations over time may be needed.

Since the analysis of a plant's behavior depends on a high number of input signals and normally deals with distributed plants and automation systems, a model-based approach is chosen in this thesis. Therefore, the main focus is on model-based approaches.

Model-based anomaly detection is the process of comparison between the real behavior of a system (via observations) and the behavior or prediction of a model (via simulation) as illustrated in Figure 3.3.

Significant work has been done in applying model-based anomaly detection to discrete event systems, i.e. changes of state values caused by the asynchronous discrete events. The finite automaton is one of the most established modeling formalisms for discrete event systems. Different types for different cases were developed (see also Section 3.2) to detect different error types. For this, finite state automata, timed automata or Petri nets are used (see e.g. [CGS07], [HZKW03], [KNJ10], [LSS01], [SLPQ06], [Tri02]).

The principle of anomaly detection consists of running the discrete-event model with observations and reporting anomalies for observations that do not fit the model. For example, in the case of finite state automata, an anomaly occurs once there is no automaton transition from the current state that should be triggered by the observed event, i.e. if the sequence of control signals (events) is incorrect. In timed automata, there are further timing constraints for the state transitions, which can be checked for timing anomalies [VBNM11].

Apart from anomaly detection in discrete event systems, there exist many approaches for the anomaly detection in hybrid or continuous systems.

Neural networks and regression-based methods have been used to approximate the functional dependency between continuous process variables and the time [VBNM11]. Sensor signals are predicted according to this functional dependency and significant deviations of the predicted signal values from the observations are reported as anomalies.

Statistical approaches to anomaly detection are predominantly based on building a probability distribution model and considering how likely objects are under that model. In most approaches, state space equations are employed for modeling the temporal transition of hidden process variables, which are related to the measurements with a measurement model. Kalman filter-based observers (e.g. [HW02], [NB07], [Hen02], and [ZKH$^+$05]) or particle filters (see e.g. [WD09]) can be used to estimate the hidden process variable. Hidden Markov Models [JKH08] assume value-discrete process variables whose temporal transitions are described with a probability matrix. Statistical anomaly detection usually comprises state estimation i.e. the estimation of the hidden process variable, residual generation and decision making. Methods like the sequential probability ratio test, the cumulative sum algorithm, the generalized likelihood ratio test or a local approach proposed in [BBN93] can be used for change detection in the residuals. In [LH11] and [SH09], a decision-making algorithm proposed in [SS80] is applied, which is based on estimates for the residual and its co-variance matrix.

Furthermore, approaches for anomaly detection are used in strictly continuous systems like neural networks, support vector machines, radial basis functions, adaptive resonance theory, qualitative trend analysis models, signed directed graphs or principal component analysis based methods are e.g. considered in [Bar10, SCZ$^+$09].

Clustering-based methods create groups of strongly related objects and find objects which do not strongly belong to any cluster [TSK06]. Examples for clustering algorithms are K-means, agglomerative hierarchical clustering, DBSCAN, Fuzzy clustering, mixture model clustering, density-based clustering, graph-based clustering or the application of self-organizing maps (SOM) (see [TSK06] for details). The

use of self-organizing maps (SOM) for anomaly detection has been investigated in [Fre08].

Industrial processes typically consist of both continuous physical processes and discrete events which are caused by discrete valves, on/off switches, logical overrides etc. [RT11]. Such hybrid systems are characterized by discrete modes with continuous process behavior and discontinuities caused by mode transitions. For this reason, anomaly detection in hybrid systems usually comprises a combination of the described methods for fault detection in an underlying discrete event system and fault detection in the strictly continuous process behavior, which can be assigned to a discrete mode. Mode transitions can be attributed to control events, continuous system variables crossing threshold values or probabilistic changes according to an underlying Markov process. Many approaches abstract from the control events and assume probabilistic mode changes or mode changes according to continuous system variables.

5.2 Identification Algorithms for Finite Automata

In Section 4.2, some identification frameworks were introduced, which are used to identify finite automata. This section refines it and gives a classification for the identification algorithms for finite automata. After this, we give an overview to identification algorithms and their functional principles for each class.

5.2.1 Classification of Learning Algorithms

Several algorithms exist to identify finite automata. They can roughly be distinguished by the following features:

- ***Offline* or *online* learning:** Offline identification algorithms presume a data acquisition and storage for further usage. They can apply preprocessing during the learning process since the data is stored and it is possible to access the data multiple times. Online learning is more real-time critical. Here, the algorithms are allowed to access the data only once. In one step, this data sample has to be included into the model. No preprocessing and storage of the data is possible.
- ***Passive* or *active* learning:** Passive learning algorithms have to cope with a given set of observations to learn the model. Additionally, passive algorithms are not able to choose the data or request additional information. In contrast, active learning algorithms can ask for additional data or further information, if needed. Active learning algorithms often imply a teacher-student relationship. It is related to the framework of query-learning.
- **Given *finite* or *infinite* data:** The available data, which is used by the learning algorithms, can be finite or infinite (c.f. identification frameworks 'identification from given finite data' and 'identification in the limit' from Section 4.2). Usually the available data is finite, this at least holds true for the application in cyber-physical production systems. Though, it was shown that the identification problem for DFAs from given finite data is NP-complete [Gol78] (c.f. Section

4.1). However, many identification algorithms converge to the (correct) target language if infinite data are available. Examples for identification algorithms which identify an automaton in the limit are ALERGIA [CO94], MDI [TDdlH00] and RTI+ [Ver10].

- ***Informant* or *text identification*:** As the name sounds, the *informant identification* assumes the existence of some expert. It can also be described as supervised learning. The learning algorithm gets positive learning examples (according to Definition 6) and negative examples (according to Definition 7). In the context of grammar learning, a positive example is a word accepted by the target language, on the contrary a negative example is not accepted. Example algorithms which learn from informant are RPNI [OG92] and ID_1DTA [VdWW09]. However, using identification algorithms in the context of cyber-physical production systems, negative examples are hard to generate, since it is impossible to enforce failures in the production systems and log them, nor is it possible to think about all possible failures and simulate them in a model (which, by the way, would have to be created manually). Therefore, the framework of *text identification* plays an important role in this thesis. It is a kind of unsupervised learning in which only positive examples are available.
- **Allowing *don't-care* states or not:** So far, this criterion was not considered for the classification of learning algorithms. Algorithms which use incoming and outgoing events and event sequences for the state equivalence check usually have don't-care states, since it may become possible to reach a state using different paths. To avoid don't-care states, the algorithm should check the state information itself (e.g. the signal vector in cyber-physical production systems). In [RLL10] and [RSLL12] algorithms are proposed, which identify finite automata without don't-care states.

Example 5. *Figure 5.2 shows an example to illustrate the difference between both versions. In the example setup and two switches S_1 and S_2 are used to switch a light.*

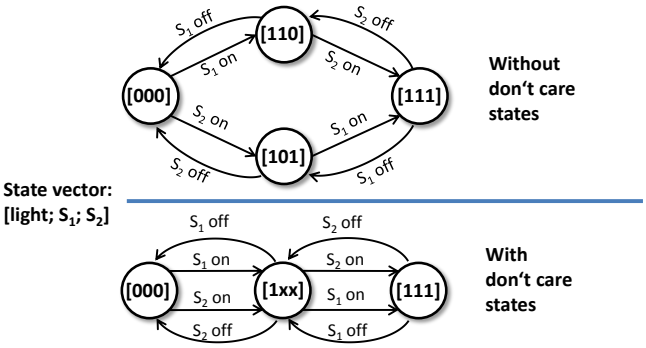


Fig. 5.2 Automata with and without don't care states.

The state vector contains three signals: the light, and the two switches. This state vector is given in each state with its corresponding value: "1" when the signal is active, "0" when the signal is not active and "x" when the signal can be active

as well as inactive. Both automata describe the same behavior, while the first automaton (not allowing don't care states) uses one state more than the second one. However, the second automaton, which contains a don't care states, is not unambiguous. Depending on the event sequence, the state in the middle can be assigned differently.

Automata without don't care states usually have more states than automata that contain don't care states.

In the literature, the terms *active* and *online* are often mixed up. The reason is that usually active learning algorithms work in an online manner and vice versa. Though, in Section 6.3, we introduce a learning algorithm that learns in an *online* and *passive* manner.

The single points of the aforementioned list do not exclude each other. Learning algorithms can belong to multiple groups, e.g. offline passive learning from text using positive examples only.

5.2.1.1 Offline passive model learning

Offline learning algorithms have to cope with a given set of observations. The data comes from a database. Most offline learning algorithms use the state merging approach to identify the structure of the automaton. The general state merging methodology is displayed in figure 5.3.

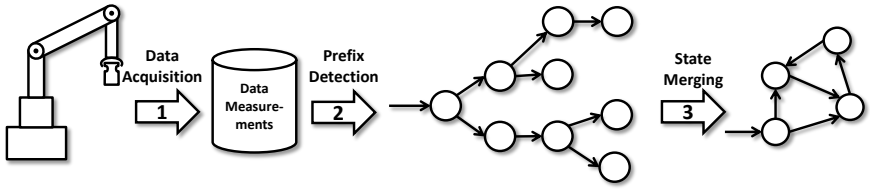


Fig. 5.3 The general offline learning methodology using the state merging approach.

In **step (1)**, the system is observed and the events are extracted. The data can also come from a simulated model, for instance a finite state machine. In both cases the data are stored into a database.

In **step (2)**, the prefix tree is created. Beginning with an initial state, a new state is created after each discrete event. The first observation sample leads to a linked list of states. The following samples begin again with the initial state. As far as the prefix is equal, the states and events are followed. When an event is observed at some point, which is not an outgoing event from the current state, a new transition and a new state are created. The prefix tree stores the event paths in a dense form since every prefix is stored only once.

In **step (3)**, each pairs of states are checked for compatibility. If a compatible pair of states is found, the states are merged. This is done to obtain a generalized automaton that abstracts the behavior of the observed system.

Several existing algorithms learn an automaton in an offline manner. They all proceed in the described steps but have different compatibility checks for state

merging and different merging strategies. The best known offline learning algorithms are *Alergia* [CO94], *MDI* [TDdlH00] and *RTI+* [Ver10]. A more detailed description to some algorithms will follow in the next subsections.

5.2.1.2 Active model learning

Active learning is also often referred to as query learning. The learning is organized as a student-teacher-relationship, where the student asks questions and the teacher answers them.

At the beginning, the learner (student) knows nothing about the language L to be identified. The teacher knows L and can answer the learners queries, which can be one of two types:

- A membership query: Is the word w accepted by the automaton A?
- An equivalence query: Is the language of the hypothesized automaton H equivalent with the language A to be identified; i.e. $L(H) = L(A)$?

The learner organizes all results of the membership queries in tables. These are used to create the automaton. The equivalence queries are used to check whether the constructed automaton is correct or not. If it is correct, the teacher answers with *yes* otherwise he returns a counterexample.

Angluin's *L** [Ang87] is one of the first and most famous active learning algorithms that identifies a DFA. The learning operates as illustrated in figure 4.3.

Grinchtein introduced an online active learning algorithm for timed target languages [GJL04] identifying event-recording automata (ERA). The algorithm works similar to L*.

5.2.1.3 Online passive model learning

To the best of our knowledge, so far no algorithm for the online passive learning of timed automata exists, which in addition gets along with only positive examples. In Section 6.3, we close this gap and introduce OTALA, which is the first online passive learning algorithm for Timed Automata using positive learning examples only.

5.2.2 Identification of untimed Deterministic Finite Automata

Algorithms which identify (untimed) DFAs use positive as well as negative learning examples. Such algorithms are often used in the field of grammatical inference. A comprehensive survey on algorithms in the field of grammatical inference is given in [dlH05]. In this thesis, only some basic algorithms are mentioned.

RPNI

One of the first identification algorithms that used the state merging approach is the *Regular Positive and Negative Inference* (RPNI) [OG92]. It identifies a DFA from informant, i.e. it uses positive as well as negative examples. It is based on the red-blue framework [LPP98]. At the beginning, the root is colored red, its descendants are colored blue, all other states are not colored at the beginning.

The red states are final, they will not be deleted. The blue states are candidates for the merging with red states. In a top-down manner the blue states are merged with the red ones. If the resulting DFA is consistent with the sample set (positive and negative examples do not end in the same state), the merge is kept otherwise rejected. Then, the descendants of the blue state are colored blue. This procedure is continued until all states are colored red.

L*

Angluin's L* [Ang87] is an active identification algorithm for the identification of DFA. As described in Section 5.2.1.2, the learning process is organized as a student-teacher-relationship, where the student asks questions (membership or equivalence query) and the teacher answers them accordingly.

5.2.3 Identification of untimed Probabilistic Deterministic Finite Automata

Identification algorithms for (untimed) probabilistic DFAs only use positive learning examples. Commonly used (passive) algorithms are e.g. MDI (Minimal Divergence Inference) [TDdlH00] and ALERGIA [CO94]. They both start with a prefix tree, go top down from the root to the leafs and search for compatible states. However, they both use a different method to compute the compatibility of states and a different state merging method.

ALERGIA

The algorithm *Alergia*, introduced by Carrasco and Oncina in 1994 [CO94], was originally developed for the identification of stochastic regular languages by using state merging techniques. It identifies a Probabilistic Deterministic Finite Automata (PDFA, see also Definition 11) using positive learning examples only. While the RPNI algorithm used the negative examples to prevent an incorrect merging, Alergia has to rely on positive examples only. So it uses the event probabilities to prevent such incorrect mergings: Two states are compatible if the probabilities of the event sequences in their respective subtrees are similar.

Alergia operates on a PTA, comparing each pairs of states in a top-down manner, beginning at the root of the tree and continuing with the states according to their lexicographical order of the shortest paths. Two states are compatible, when their arrivals, terminations and outgoing transitions belong to the same probability distribution. For this, a threshold given by the Hoeffding bound [Hoe63] is used. The children of the regarded states have to fulfill this condition as well, that is, we recursively ensure that the subtrees of two states, and the very nodes themselves are compatible. After finding two compatible states, they are merged together, which can result in a nondeterministic automaton, as the yielding state would have two transitions triggered with the same symbol. That means, we have to recursively merge subtrees of the same trigger symbols in order to reestablish a deterministic automaton. This is done without further checking for compatibilities, as their compatibility was ensured by the recursive check that had been done beforehand.

MDI

The MDI (Minimal Divergence Inference) algorithm was introduced by Thollard et al. [TDdlH00] and operates in the same problem domain as Alergia, but with a different merging criterion. MDI tries to "globally control the level of generalization from the learning sample". The creators of the MDI algorithm state that Alergia's main problem is that its state merging operation operates locally only, that is, the overall divergence of the algorithm to the sample set is not taken into consideration. MDI however, calculates the Kullback-Leibler divergence between the original PTA, and the modified PTA after each merging operation. The merging operation is only accepted if the calculated divergence is below a given threshold value, otherwise rejected again. That is, when applying the MDI algorithm on a PTA, the divergence constraints are constantly checked during runtime, which is supposed to result in an output of better quality. Note, that the global calculation of the compatibility leads to an increased calculation effort, since in each step (compatibility check of two states) two states are merged and the resulting PTA is compared with the previous one, and if the new PTA differs from the previous too much, it is rejected again.

In [VMN13], we performed a comprehensive evaluation on the performance of these and other identification algorithms.

5.2.4 Identification of Deterministic Timed Automata

The field of identification of timed automata is less mature than the identification of untimed automata. Only few algorithms have been developed so far. Most research work in this field has been done by Verwer [VdWW07, VdWW06, VdWW08, VdWW09] for the identification of Timed Automata and Grinchtein (e.g. in [GJL04]) for the identification of Event Recording Automata, a subclass of Timed Automata.

This section is not about the identification of Timed Automata according to Definition 13 only, but also about automata that can handle with timing information in general.

Verwer et. al. already presented several algorithms for the identification of timed automata. Two of them (ID_1DTA and RTI+) are mentioned here and explained roughly, since in this thesis we refer to them. Some of the algorithms require positive as well as negative examples (e.g. ID_1DTA). Two states are merged when there exists no negative example that prevents this action, i.e. this new merge would lead to an automaton, which would accept an example from the set of negative examples. Using only positive examples for the identification (as RTI+ does), some probabilistic information have to be used to determine the consistency of the automaton with the sample sets for each merge step.

ID_1-DTA

In [VdWW09], Verwer presented the algorithm ID_1-DTA for the identification of 1-DTAs. It identifies 1-DTAs efficiently in the limit from informant, i.e. it uses positive as well as negative learning examples. It is an offline leaning algorithm but does not use the state merging approach. It identifies the automaton structure piece by piece. In each identification step, one transition is identified. For this, first a data sample has to be chosen. Then for each data sample, three elements have to

be identified: *(1)* the smallest consistent lower bound for the clock guard *(2)* the clock reset and *(3)* the target state.

In [Ver10], Verwer proves that:

- given any input sample S, ID_1-DTA returns a 1-DTA A in polynomial time that is consistent with S and
- if S contains a characteristic subsample S_{cs} for some target language L_t, then ID_1-DTA returns a correct 1-DTA A.

That is, the algorithm ID_1-DTA identifies 1-DTAs efficiently in the limit.

RTI+

In [Ver10], Verwer gives an algorithm, *Real-Time Identification from positive data*, RTI+, which identifies a PDTA (with one clock) from text, i.e. it identifies a PDTA using positive examples only. Verwer proved that RTI+ identifies PDTA in polynomial time in the limit. RTI+ is an extension of the algorithm RTI [VdWW07], which identifies a non-probabilistic DTA from informant, i.e. using positive and negative examples. Verwer denotes the formalism as probabilistic deterministic real-time automaton, PDRTA, which is a probabilistic DTA with one clock, and the clock is reset in each transition.

The general identification method is illustrated in Figure 5.3. Since it uses the state merging approach, first, a timed prefix tree acceptor is created, which is generally the same as the untimed PTA but additionally contains time information.

In the second step, applying the top-down strategy, i.e. beginning with the root and proceeding to the leafs of the PTA, the algorithm checks whether there exist compatible states that can be merged. Additionally a splitting process is introduced. A transition can be split if the resulting sub trees are significantly different.

The red-blue-framework [LPP98] is used to mark states in red and blue. In each iteration step, a blue state is selected, which is visited most often and all possible merges of this blue state with all red states and all possible splits of the incoming transition of the blue state are evaluated and the p-values are calculated. If the lowest p-value of a split is less than 0.05, the split is performed, otherwise, if the highest p-value of a merge is greater than 0.05 then this merge is performed. After a split is performed, the prefix sub trees from the subsequent nodes have to be renewed. The decision whether two states have to be merged or a transition has to be split is made on the p-value, which is calculated using the likelihood ratio test.

5.2.5 Identification of subclasses of Deterministic Timed Automata

Apart from the identification of Deterministic Timed Automata, there exist approaches which deal with subclasses of Timed Automata. Some of them are mentioned in this subsection. Since they don't play a big role in the context of this thesis, they are explained only roughly.

Identification of Event-Recording Automata (ERA)

Event-Recording Automata belong to a subclass of Timed Automata. The main difference is that ERA measure the time between the occurrence of single events, i.e. the time is measured for each event until it occurs the next time. Details to the formalism of ERA can be found in [AFH99].

Grinchtein et. al. [GJL04] developed an active algorithm for the identification of ERA. It extends Angluin's algorithm L* for active learning of regular languages to the setting of timed systems. The general method follows the principle of active learning as explained in Section 5.2.1.2.

The first algorithm (from Grinchtein) needed an exponential amount of queries. In [LAD+11], the algorithm TL* is introduced, which is an efficient extension of Angluin's algorithm L* and Grinchtein's algorithm. They prove that TL* is correct and terminates in a finite number of iterations and that the ERA learned by TL* has the minimal number of locations.

Identification of Finite Automata based on state vectors

A completely different approach is proposed in [RLL10] and [RSLL12]. The approach does not stem from the field of grammatical inference. Instead it is more application-oriented. The application scenario is equal to this thesis, the anomaly detection in cyber-physical production systems. They use a formalism, which they call *non-deterministic autonomous automaton with output (NDAAO)*. The main difference to timed automata is the definition of the state information using a vector with the values of the given signals (inputs and outputs, I/O-vector). The learning procedure is based on the assumption that each I/O vector is created by a new event.

In the learning procedure for each incoming observation (which is an I/O vector), it is checked whether it was observed before. If this is the case, it is also checked whether an appropriate transition exists. If a state or transition does not exist, it is created. The transitions don't contain an event and they are triggered by time only, which is expressed as a time range with the minimum and maximum time stamp.

Identification of Continuous-Time Markov Chains

A slightly different yet closely related field is addressed by Sen et. al. [SVA04]. They use Edge Labeled Continuous-Time Markov Chains (CTMC_L) as formalism for timed behavior models. The identification algorithm is based on the state merging paradigm introduced in RPNI [OG92] and ALERGIA [CO94]. They construct a prefix tree and then look for compatible states and merge them. It also uses the same order. The only difference is the usage of an additional condition for the amount of time spent in a state, which is based on the Chebychev inequality.

5.2.6 Identification of Probabilistic Deterministic Hybrid Automata

The aforementioned algorithms only consider discrete events in the transitions. As shown in Figure 5.4, Learning continuous signals as discrete events in a discrete Automaton would lead to a huge amount of states since every change of any signal corresponds to an event. A more effective way is to use hybrid automata in which only discrete signals describe events and continuous signals are approximated in the states.

The first algorithm which learns a Hybrid Automaton is called HyBUTLA: Hybrid Bottom Up Timing Learning Algorithm [VKBNM11]. It was developed in cooperation with this thesis. HYBUTLA is based on BUTLA. While BUTLA identifies

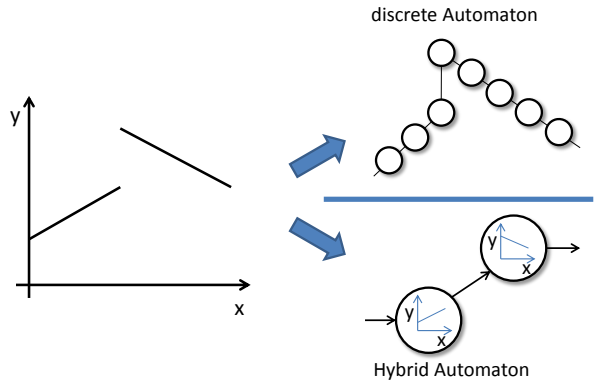


Fig. 5.4 The idea and operating principle of hybrid automata.

the automaton structure and learns the timing behavior, HyBUTLA includes the continuous behavior into the states. If necessary, the states are also split. The decision is based on an autonomous jump detection using the wavelet transform for example. This enables the hybrid automaton to generate transitions that are not only based on discrete events but also on crossing a threshold or on sudden changes in a continuous signal, which is dependent on time.

5.2.7 Identification of Nondeterministic Automata

Only few approaches deal with the identification of Nondeterministic Automata. The **DeLeTe** and **DeLeTe2** algorithms were introduced by Denis et. al. in [DLT04] for the identification of Residual Finite State Automata (RFSA), which is a subclass of Nondeterministic Automata. Since the identification of Nondeterministic Automata is not subject of research in this thesis, it is only referred to the literature.

5.3 Conclusion

In this part, we gave an overview to timing modeling formalisms and argued why we are focusing on finite state machines and, more specifically, on timed automata (Chapter 2). The main reasons are the simplicity, the fact that they can be identified from observations only and their applicability for anomaly detection.

In Chapter 4, we worked out that our identification framework is the *identification in the limit*. Several state of the art identification algorithms have been presented in Section 5.2. RTI+, so far the only existing algorithm that identifies a timed automaton from text, has three main disadvantages:

- The top-down strategy requires a (time consuming) recursive compatibility check.
- The evaluation of all possible merges and splits additionally consumes time.

- After every performed split, the prefix tree acceptor has to be renewed from the corresponding transition, this again consumes time and, additionally, a lot of space to store all the necessary information.

To overcome these disadvantages, a new identification algorithm for timed automaton will be presented. It introduces a different timing learning method and a new state merging strategy.

As already mentioned in Section 5.2, to the best of our knowledge so far no identification algorithm exists that identifies a Timed Automaton in an online and passive manner. But this is required for use in cyber-physical production systems. To close this gap, an appropriate identification algorithm is presented.

Part II

Algorithms and Theory

Chapter 6

Algorithmic Results

This chapter presents the main contribution of this thesis: Algorithms for the identification of Timed Automata. A special focus is on the suitability of the algorithms for the identification of the normal behavior in cyber-physical production systems. The identified automata are finally used for anomaly detection.

The main contributions of this chapter are the following:

- We describe how events are generated from timed observations in production plants.
- We give a new offline passive identification algorithm BUTLA, which is the first algorithm that uses the bottom-up merging strategy.
- We describe a new method to identify the timing behavior that avoids the time consuming splitting operation.
- We introduce a new online passive identification algorithm OTALA, which is the first online passive identification algorithm for timed automata.
- We present the anomaly detection algorithm ANODA.
- We give algorithms for the adaptive identification for both, BUTLA and OTALA.

The structure of this chapter is organized as follows: In Section 6.1, the requirements are analyzed, which are used for the modeling formalism and the learning algorithm. Section 6.2 introduces the offline passive algorithm BUTLA. It firstly introduces the bottom-up merging strategy, which results in a computation speed increase. Additionally it uses a different timing learning approach. In Section 6.3, the online passive identification algorithm OTALA is introduced, which to the best of our knowledge is the first online passive identification algorithm for timed automata. It is especially created for the identification of rather small stand-alone embedded devices or cyber-physical production systems. Section 6.4 presents the final usage of the identified models: the anomaly detection. The algorithm ANODA is presented, which uses the identified timed automata to find anomalies in production systems during runtime. Finally, in Section 6.5 we show that how the models can be adapted during runtime when an error is signaled, whereas it belongs to the normal behavior.

This chapter is partially based on contributions which we published on international conferences [MNV+11, VKBNM11, VBNM11, MKPGN13, Mai14].

6.1 Requirements on modeling formalism and identification algorithm

The identified model shall be used for anomaly detection in production plants in the automation industry. Therefore, the modeling formalism and the identification algorithm should be adapted to this use case. In this section, the main requirements are listed.

Requirements on the modeling formalism:

In general, model-based diagnosis can use any kind of behavior models. However, the quality of diagnosis depends on the used modeling formalism and the prediction abilities of the models. In this section, we give some requirements on this formalism for the use case of anomaly detection for production plants.

- *State based systems:* Production plants mainly show a state based behavior, i.e. the system's state is precisely defined by its discrete IO signals.
- *Consideration of time:* Since actions in production plants essentially depend on time, the formalism has to consider it as well.
- *Probabilistic information:* Here, the behavior models describe the previous, recorded plant behavior. So unlike in specification models, behavior probabilities must be modeled.
- *Distributed systems:* Most production plants consist of several (distributed) modules. The formalism has to deal with parallel components.
- *Hybrid data:* Most production plants contain not only discrete signals but also continuous ones. So an appropriate formalism has to consider both, i.e. the identified model should be hybrid (comprising both, discrete and continuous behavior). This aspect is considered in [Vod13], so we only consider a discrete behavior in this thesis.
- *Learnability:* Since the goal is not to create the model manually, it should be possible to identify the model automatically using an identification algorithm.

Requirements on the identification algorithm:

In principle, all of the mentioned requirements from above (requirements on the modeling formalism) also apply to the identification algorithm. The identification algorithm should be able to capture the following issues:

- State based behavior
- Timing information
- Probabilistic information
- Distributed systems

Furthermore, the following issues have to be considered:

- *Parallelism structure:* Based on a given parallelism structure, the identification algorithm should be able to identify independent (parallel running) components.
- *Hybrid data:* This requirement is captured in [Vod13], where the algorithm HyBUTLA is introduced. HyBUTLA is based on BUTLA, which is introduced in Section 6.2.

Requirements on the system to be identified:

Finally, there are some requirements on the system for which the behavior has to be identified.

- *Observability:* In general, only the observable effects can be identified. Therefore, the system and its behavior have to be observable. Partially, this also applies to the anomaly detection. Anomalies in observable signals can be detected reliably. However, some anomalies can be detected indirectly, e.g. the wear of a conveyer belt can be detected using the time it needs to convey something from one point to another.
- *Deterministic behavior:* The system should behave in a deterministic way, since the chosen modeling formalism can not handle non-determinism. Each non-deterministic situation is determinized again in the model.

6.2 Bottom Up Timing Learning Algorithm (BUTLA)

This section introduces the *Bottom Up Timing Learning Algorithm* (BUTLA). It only uses positive learning examples to identify the automaton structure, including states and transitions. The transitions between two states contain information about the timing constraints as relative time stamps referred to the entering of the state and probabilistic information. The identified automaton is a one-clock Probabilistic Deterministic Timed Automaton according to Definition 15.

BUTLA differs from existing automata identification algorithms (see Section 5.2) in the following points :

- **Bottom-up learning strategy:** All existing offline identification algorithms following the state merging approach, such as *Alergia* [CO94], *RPNI* [OG92], *MDI* [TDdlH00] or *RTI+* [Ver10] (see also Section 5.2) use the top-down strategy to search for compatible states. Beginning with the root of the prefix tree acceptor, all pairs of the subsequent states are checked for compatibility using the lexicographical order of the shortest path. BUTLA is the first algorithm that introduces the bottom-up strategy. Beginning with the leafs of the prefix tree acceptor all previous states are checked for compatibility using the lexicographical order of the longest path. This bottom-up strategy works best if all leafs of the prefix trees correspond to final states or to the same states in a cyclic process. Here, the new algorithm applies the domain specific knowledge: For measurements of cyber-physical production plants, it is usually not a problem to guarantee this constraint.
  A detailed evaluation of the advantages and disadvantages of both strategies follows in Section 7.4.
- **New timing learning:** The timing information is given as relative time stamp from the last occurring event and is usually modeled as time span, including the minimum and maximum value of the recorded time values for a certain transition. BUTLA additionally uses a probability density function (PDF) over time, which reflects the system's behavior in more detail. In contrast to other identification algorithms (e.g. RTI+ [Ver10]), the timing identification procedure directly refers to the PDFs to determine whether two timed events belong to the same process. Further details are given in Section 6.2.3.
- **Merging criterion:** The merging criterion basically uses the Hoeffding Bound similar to the algorithm Alergia [CO94]. However, this criterion has to be adapted since BUTLA uses the bottom-up strategy. Instead of using the out-going transitions as Alergia does, BUTLA considers the incoming transitions to calculate

the compatibility criterion. As to the rest, the determinization process is equal to those algorithms. Details will follow in Section 6.2.5.

- **Avoiding split operations:** The best known algorithm for learning *timed* automata (RTI+ in [Ver10]), adds an additional splitting step to the merging procedure mentioned above: transitions are split by subdividing the corresponding time interval. The split is performed when the resulting new sub-trees do not resemble each other. Besides the runtime problems (this is done for all possible splits of a time interval), this procedure does not necessarily take natural clusters in the timing information into consideration. BUTLA avoids this splitting operation by including an additional preprocessing step that identifies (multiple) timing distribution functions and uses this information in an early stage such that a splitting operation becomes dispensable. A more detailed evaluation is given in Section 7.5.

The identification algorithm BUTLA follows the methodology from Figure 6.1:

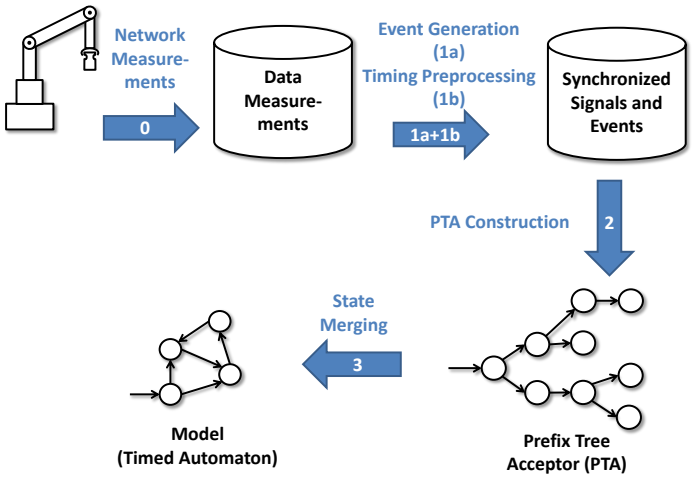


Fig. 6.1 BUTLA identification method.

- **Step 0: Network measurements**
  First of all, all relevant data is measured from the system. For this, the system is observed during several production cycles. The resulting observation sequences (recorded signal vectors and time stamps) are stored in a database.
- **Step 1a: Event generation**
  An event is defined as the change between two subsequent signal vectors (see Definition 3). Thus, the events are generated by extracting the differences between the signal vectors based on the recorded signal vectors and time stamps. The event's timing is calculated as relative time value to the previous event.
- **Step 1b: Timing preprocessing**
  Then, the timing of the events is analyzed in a preprocessing step. The relative time values of each event are collected in a histogram. It is decided whether the timing behavior is subdivided into multiple modes based on this histogram and

the resulting probability density distribution over time. In case of multiple modes, an event is separated according to the number of modes in the PDF such that each event consists of only one mode. For instance, an event e_i from the alphabet Σ with 2 modes is separated into $e_{i,1}, e_{i,2}$ forming Σ', an extended set of symbols, as can be seen in Figure 6.2. In the following steps (construction of the prefix tree acceptor and state merging), only this modified alphabet Σ' is used.

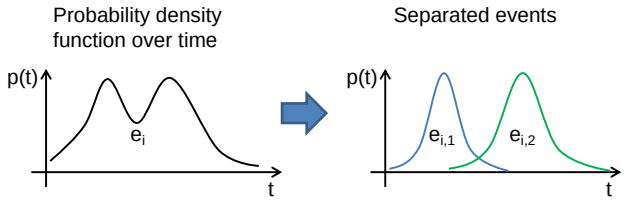


Fig. 6.2 An event with a multi-mode timing behavior is separated into its modes.

- **Step 2: Construction of the Timed Prefix Tree Acceptor**
  In the next step, common prefixes of data sequences are detected. For the first cycle, the sequence of events is stored in form of a linked list of events. Then for each following cycle, common prefixes with a previous event sequence are detected; if the actual sequence derives at some point, it leads to a new branch. The final result is a prefix tree (prefix tree acceptor, PTA), which models all observation sequences in a dense form—dense because common sequences are stored only once.
- **Step 3: State merging**
  This step is the core of the identification algorithm. Now, similar states of the prefix tree acceptor are merged. Using the bottom-up strategy in the lexicographical order of the longest paths, all pairs of states are checked for compatibility. If a pair of states satisfies the compatibility criterion, the states are merged. The structure of the automaton is identified using the state merging method. Note that unlike other identification approaches, this structure does not have to be given but is identified based on observations. The result is a Timed Automaton, which represents the timing behavior of the observed system in an abstracted way.

A detailed description for each individual steps is given in the following subsections.

The complete algorithm *BUTLA* is shown in algorithm 1 and works as follows: Learning examples according to Definition 2 are given as input. Details on the data acquisition are given in Section 6.2.1.

First, the events are generated (line 1, details in Section 6.2.2). Then the timing is preprocessed (line 2, details in Section 6.2.3), creating a new set of symbols Σ'. Then, a prefix tree is created based on the generated events and the new set of symbols (line 3, details in Section 6.2.4).

Finally, compatible states are merged in a bottom-up order (line 4-9); Each pair of states is checked for compatibility (line 5). How state compatibility is defined will be explained in Section 6.2.5. If a pair of states is found to be compatible, they are merged into one state (line 6). The new state obtains all incoming and outgoing transitions from the compatible states and the state leaving probability is averaged.

After each merging step, the resulting automaton can be non-deterministic. So the automaton has to be determinized (line 7). Recursively, it is checked whether there exist two outgoing transitions with the same symbol for each subsequent state. If this is the case, then both descendents are merged. Note, that the modified set of symbols Σ' is used here and therefore it may happen that two different symbols contain the same event but stemming from different time modes. This has been separated in the timing preprocessing.

The final result is the identified Timed Automaton according to Definition 15.

Algorithm 1 Bottom-Up Timed Automata Learning Algorithm BUTLA.

Algorithm BUTLA (Σ, $\mathbf{D}$):
Given:
(1) Alphabet Σ
(2) Observations $\mathcal{D} = \{\mathbf{D}_1, \ldots, \mathbf{D}_n\}$ where $\mathbf{D}_i \in (\Sigma \times \mathbf{R})^*$,
$\mathbf{D}_i$ is one sequence of timed events (e.g. a system cycle, according to Definition 2)
Result: Timed Automaton $\mathcal{A}$ (according to Definition 15)
(1) Compute events Σ based on $\mathcal{D}$.
(2) Σ' = timingPreprocessing(Σ)
(3) Build prefix tree $PTA = (S, s_0, F, P, \Sigma', E)$ based on $\mathcal{D}$.
PTA is a timed automaton according to definition 15.
(4) **for all** $v, w \in S$ in a bottom-up order **do**
(5) **if** compatible(v, w) **then**
(6) $\mathcal{A}$ = merge(v, w)
(7) determinize($\mathcal{A}$)
(8) **end if**
(9) **end for**
(10) **return** $\mathcal{A}$

6.2.1 Step 0: Network measurements

A prerequisite for model identification in cyber-physical production systems is the data acquisition. Different methods exist to obtain the required data: *(1)* direct connection to the PLC, *(2)* a network tap to sniff the data, *(3)* a mirrored port on a network switch, *(4)* writing the data directly from the Supervisory Control and Data Acquisition (SCADA) system into a database or *(5)* special data acquisition cards in parallel to the IOs in the plant. In this thesis we stick to the second solution. The principle of the data acquisition is shown in Figure 6.3. Basically, the data is acquired from the network using a network tap. A special data logger extracts the payload out of the data stream.

The data, which is acquired in the production plant, is stored according to the Definition 2 as a matrix with timestamps and the associated signal vector:

$$D = \begin{bmatrix} t_1 & q_{1,1} & \cdots & q_{d,1} \\ t_2 & q_{1,2} & \cdots & q_{d,2} \\ \vdots & \vdots & \vdots & \vdots \\ t_l & q_{1,l} & \cdots & q_{d,l} \end{bmatrix},$$

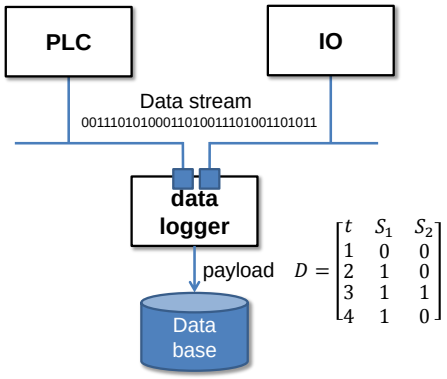


Fig. 6.3 The principle of data acquisition using a network tap.

The first column represents the time stamps in an absolute time base beginning at the start of the measurements. The other d columns represent the signal vectors to each time stamp.

The observed system can consist of several independent subsystems. However, BUTLA is not able to identify these subsystems and to subdivide the overall behavior into single independent components. This also applies to almost all identification methods and algorithms. Therefore, this has to be done beforehand. For this, several methods exist, of which two are roughly explained here:

- **Approximation of the parallelism structure using the system topology**
  We use one method (introduced in [MNV+11] and [JJN11]), that is especially suitable for distributed production plants connected via Profinet. It is based on the assumption that each independent subsystem is connected to one separate IO module. In practice, this mostly works well, since signals which belong to one subsystem are mostly located close to each other locally. For this, the topology of the automation system is used to approximate the parallelism structure. This parallelism structure decomposes the overall model into parallel components—for which sequential behavior models can be identified.
  AutomationML [Aut14] can be used as an exchange format to store the topology of the automation systems — and therefore the parallelism structure. This parallelism structure includes information about the IO devices, Programmable Logical Controllers (PLCs) and communication networks.
  Here, the Profinet standard is used as an example. The result is an unsorted collection of all participants in the network.
- **Automatic identification of the parallelism structure**
  In his dissertation, Roth describes a method that how such a parallelism structure can be identified automatically based on recorded observations [Rot10]. He uses the term *partitioning*. The partitioning of discrete event sub-systems can be obtained using expert knowledge, but also by analyzing the event sequences. In the latter case, the event sequences are analyzed concerning the correlation and concurrency of subsets of the event sequences. For this, some metrics are used (e.g. introduced in [MWvdBD03]), which allow to compare two arbitrary events and to obtain information about their concurrent or causal relation.

Based on the identified parallelism model, which includes information about independent sub-modules, the algorithm BUTLA can be used to identify the timed behavior models as described in the following subsections. For each independent sub-module, a separate model is identified based on its associated input and output signals.

6.2.2 Step 1a: Event Generation

The network measurements are used to generate the alphabet and the timing constraints. Inputs are positive samples according to Definition 2. The events e are defined as changes between two subsequent signal vectors and the resulting time stamp (according to Definition 3).

In the first step, these changes are detected. Furthermore, the time stamps are calculated. Since only one clock is used, the only possibility is to use relative time, i.e. to reset the clock after each event. To obtain the relative time stamps, the difference between each occurring events is calculated.

Example 6. *Let us consider an example measurement D_i with 4 discrete signals a, b, c, d.*

$$D_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 27 & 1 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 42 & 1 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 77 & 1 & 0 & 1 & 1 \\ 84 & 1 & 0 & 1 & 1 \end{bmatrix},$$

First, the events are generated based on the difference between two subsequent signal vectors. Here the events are the following (as time value, first the absolute time stamp is captured):

e_1*:'b=1' (t=3)*
e_2*:'a=1' (t=27)*
e_3*:'b=0' (t=42)*
e_4*:'c=1;d=1' (t=77)*

Then, the relative time stamps are calculated, as discussed in our example:

e_1*:'b=1[3]'*
e_2*:'a=1[24]'*
e_3*:'b=0[15]'*
e_4*:'c=1;d=1[35]'*

Note that only changes of the values in discrete signals generate an event. It may happen that equal signal vectors appear consecutive (e.g. between 77 and 84 time units in the example data matrix). This does not lead to an event. The same applies to

present values of continuous signals (not displayed in the example data matrix). The latter case is considered in the partner dissertation [Vod13], extending BUTLA to the identification of Hybrid Timed Automata → HyBUTLA.

Now, all samples D_i from the sequence $D = \{D_1, D_2, ..., D_n\}$ are used to generate the events. Additionally, a list of observed time values is created for each event. The events are now available in tupels:

$$\{e_i, \{t_1, t_2, ..., t_k\}\},$$

with e_i a certain event, $k \in \mathbb{N}$ the number of observations with this event and $\{t_1, t_2, ..., t_k\}$ is a list with the observed time values.

The listing of all events is stored in Σ. It is the alphabet of the automaton.

6.2.3 Step 1b: Timing Preprocessing

Once the data is stored in a database, the timing behavior of the observed system is analyzed. BUTLA firstly introduces a preprocessing step to identify the timing behavior of events. In the context of this section, timing behavior means how timing is represented for specific events in the automaton.

Several methods can be used to describe the timing behavior in events:

1. Time ranges: Only the minimum and maximum time values are used to determine the time bounds.
2. Uni-modal timing distribution functions: All time values are supposed to stem from the same (uni-modal) distribution function.
3. Multi-modal timing distribution functions: It is considered that a mixture of uni-modal timing distributions can also occur. All single uni-modal distribution functions are identified. A uni-modal distribution function is a special case of multi-modal distribution functions.

These methods are described in the following subsections.

6.2.3.1 Timing Identification based on Time Ranges

The identification of the timing behavior based on the time ranges is the most simple version. Here, it is just mentioned for the sake of completeness. The identification is rather simple: For each transition, the minimum and maximum time value is stored. Additionally, no timing preprocessing is necessary.

However, a timing model with only minimum and maximum time values is not expressive enough since it does not necessarily represent the timing behavior in a proper way. This can be seen in Figure 6.4: Obviously there are two clusters of timing values but this information can not be captured by identifying only the minimum and maximum time value.

Furthermore, the bounds are too strict from the anomaly detection point of view. This can also be seen in Figure 6.4. In case of normal operation, the time value is in the green area. An error will be signaled while crossing the border to enter the red area. Warnings can not be issued due to the strict borders.

Fig. 6.4 Strict borders with minimum and maximum time stamps.

6.2.3.2 Unimodal Timing Distribution Function

As already mentioned, BUTLA uses a different time learning operation. Here, we use a different heuristic to learn the correct timing information at the transitions. The timing is expressed by means of probability density functions (PDF) instead of time intervals; this allows for a much preciser model of the timing.

Especially from the anomaly detection point of view, the usage of PDFs is advantageous. In Figure 6.5, it can be seen that it gives a warning before signaling an error. The green area describes the normal behavior. If a time stamp enters the yellow area, a warning will be given and if it enters the red area, finally an error will be signaled.

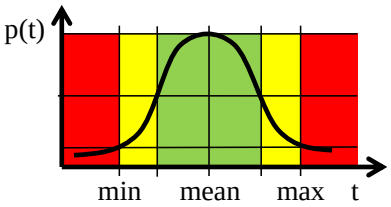


Fig. 6.5 The usage of probability density functions over time allow to give warnings before signaling an error.

In this subsection, we first assume that the timing behavior can be described using a uni-modal distribution function and we analyze which distribution function is suited best to describe the timing in transitions. Multi-modal distribution functions/ mixture models are covered in the next subsection.

When talking about probability density functions, the question about the appropriate distribution function arises. Here, we consider timed processes of technical systems. The timed events which we observe are a mixture of randomized events. This leads to the assumption that the Gaussian distribution function should be a good choice. However, the established Gaussian distribution function also has its counter-argument. Since the Gaussian distribution function is defined over $[-\infty, \infty]$, there exist positive probabilities for negative values. However, from timing modeling point of view, negative time values are impossible. This effect can be eliminated by additionally using time ranges with the minimum and maximum observed time stamp.

We performed some experiments in real plant data sets to discover the most appropriate distribution function. For this, we have acquired some data from a real production plant and we have calculated the empirical accumulated density function for the time values of each event. Then we used the the Kolmogorov Smirnov test

(see e.g. [EDJ+71] for details) for each event to calculate the p-value. The larger the p-value, the more similar are the chosen distribution functions, particularly the empirical data from the observations and one of the reference distribution function.

Figure 6.6 shows some typical cumulated density functions (CDF) for one event of a test data set and chosen distribution functions. Although it is just a snapshot of one event, it can be seen that the Gaussian distribution fits the test data set best. Other events in the data set performed similarly.

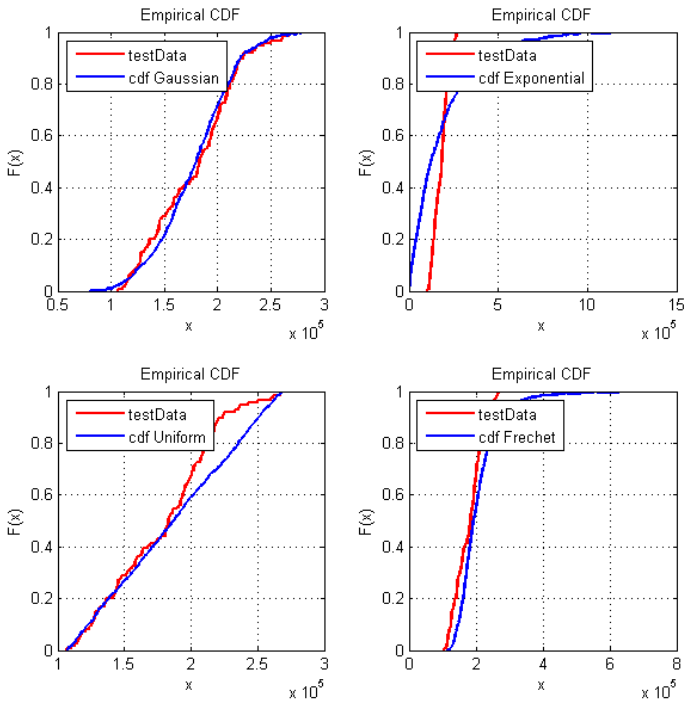


Fig. 6.6 Cumulated density functions for a data set and chosen distribution functions.

The box plot in Figure 6.7 gives an overview about the calculated p-values for each distribution function. It can be seen that the Gaussian distribution by far gives the best results using the p-value.

Furthermore, in Table 6.1, the numbers that belong to the distribution function are given. A data set is fitting to a given distribution function if the calculated p-value is larger than 5%. It can be seen, that the Gaussian Normal Distribution is most suited to describe timing in technical processes.

Since the Gaussian distribution gives the best results, in the following subsections, we do not use the other distribution functions.

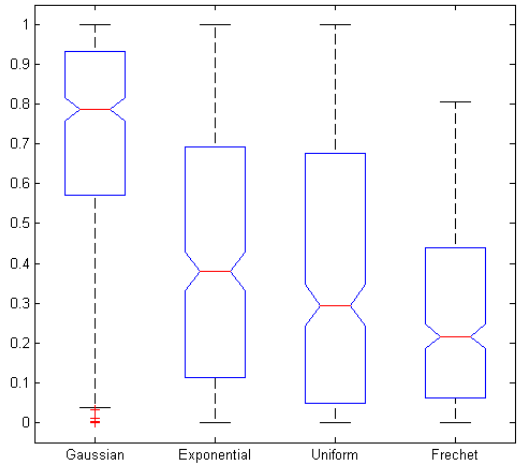


Fig. 6.7 Box plot for calculated p-values of Kolmogorov Smirnow Tests.

Table 6.1 Number of fitting functions for each chosen distribution function. The fitting is counted if the calculated p-value is larger than 5%.

	fit	don't fit	% fit
Gaussian	344	8	97.7%
Exponential	282	70	80.1%
Uniform	263	89	74.7%
Frechet	271	81	77.0%

6.2.3.3 Identification of multiple distributions in events

Often, some events (e.g. switching on a conveyor belt) appear several times in the set of observations. In that case, it must be checked whether these events are generated by the same process; this is done based on the events' timing. As described in Definition 15, an event's timing is defined as the relative time span since the last event occurrence. So, for each available event a, the Probability Density Function —probability over time—is computed. If the PDF is the sum of several Gaussian distributions, separate events are created for each Gaussian distribution.

As shown in Figure 6.8, the signal a controlling the robot is used for two different processes with two different timings (i.e. PDFs): Containers are sorted according to their size, each size results in a different timing of the robot movements.

One decision lies at the core of transition timing learning: Should a transition with an event e be split into two transitions with different timing information? Unlike other approaches, we base our decision on the timing information itself, not on the sub-tree resemblance. Figure 6.9 shows an example:

s_0 and s_1 are two states in an automaton, the transition timing is a statistic for the transition occurrences in the past and is expressed as a probability density function (shown next to the transition).

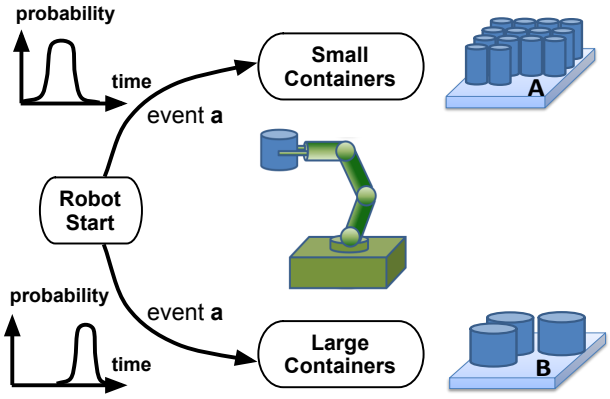


Fig. 6.8 An event which is used in different process contexts is treated as two different events.

Using Verwer's approach [Ver10], the transition would only be split (new states s_1' and s_1'') if the new resulting sub-trees are significantly different. The motivation is that different states should define different successive behaviors, i.e. sub-trees.

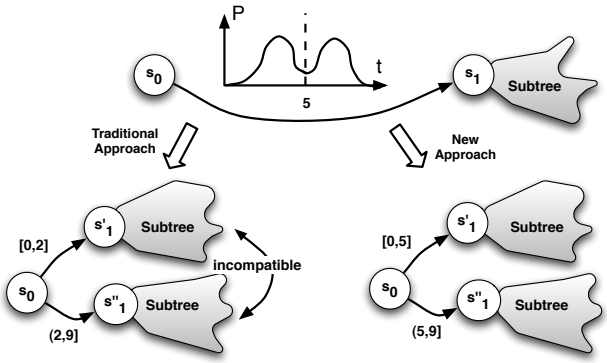


Fig. 6.9 A different timing learning approach.

But looking at the transition's timing in the figure, a split could be justified just on the basis of probability density functions: Obviously the density function is created by two overlapping Gaussian distributions. So it can be presumed that two different technical processes have created the corresponding event—i.e. here again we apply domain specific knowledge. Different processes must be modeled as different states, because only then the learning algorithm can associate transitions with the correct timing and only such a precise timing association allows for a correct separation between correct and erroneous behavior (anomaly detection).

So far, existing identification algorithms (see Section 5.2) do not consider time while creating the prefix tree, i.e. initially the data set is considered to be untimed and the prefix tree is constructed. The timing information is included afterwards. This

gives the need of additional operations like the splitting of transitions, because the knowledge about the timing information is not available.

To avoid the splitting operation, the timing modes have to be identified in a preprocessing step. The method for detecting multiple modes in events roughly works as follows (see also Figure 6.10): For each event, a probability density function (PDF) over time is calculated based on the list with the observed time values. If the resulting PDF is multi-modal, i.e. a sum of (overlapping) individual distributions, the event is separated into these modes according to the number of identified modes.

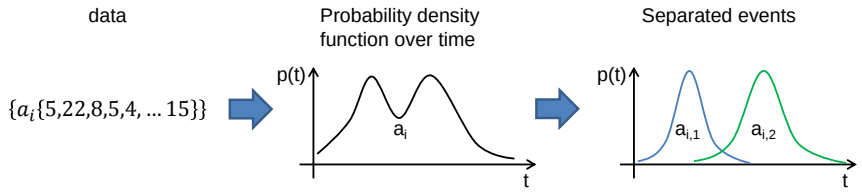


Fig. 6.10 Principle of the mode separation method.

For the detection of multiple modes in events, three methods have been evaluated:

- *Kernel density estimation:* This version is straight forward by estimating the density of the distribution function and subdividing at local minimums. It is optimized for efficient computation time. Nevertheless it delivers useful results.
- *EM - algorithm:* This method is well-known from the state of the art. It performs well, but the number of mixed distribution functions has to be known or determined subsequently by trying all values and take the best fitting.
- *Variational Bayesian inference:* This version needs the most time but delivers the best results. The number of overlapping distribution function is calculated in an iterative manner.

Due to the high computation effort of the EM-algorithm and Variational Bayesian inference, we chose to use the Kernel density estimation for the timing preprocessing in BUTLA. Since the kernel density estimation is used in the following, it is described in more detail.

Algorithm 2 shows how the kernel density estimation is used to detect multi-modal timing distributions and how multiple-mode distributions are separated into single-mode distributions.

The algorithm is about the generation of an extended set of symbols. Therefore, a new set Σ' is initialized as an empty set in line 0.

First, in line 1, all timing values $t_1, t_2, ..., t_k$ are collected and stored in a list $\{a, \{t_1, t_2, ..., t_k\}\}, k \in \mathbb{N}$ for each event $a \in \Sigma$.

Then, in lines 2-3, the PDFs are calculated using the kernel density estimation method for each event $a \in \mathbb{N}$. Density estimation methods use a set of observations to find the subjacent density function. Given a vector $\mathbf{t}$ with the time values of the observations, the underlying density distribution for a time value t can be estimated as

Algorithm 2 Timed preprocessing algorithm with Kernel Density Estimation.

Given:
(1) Measurements $\mathcal{D} = \{\mathbf{D}_0, \ldots, \mathbf{D}_{n-1}\}$ where $\mathbf{D}_i$ is one sequence of events over time according to Definition 2 (i.e. one measurement or one scenario).
(2) set of symbols Σ (from Definition 15)
Result:
extended set of symbols Σ'
(0) $\Sigma' = \emptyset$
(1) Collect all relative time values for each event $a \in \Sigma$
(2) **for each** $a \in \Sigma$
(3) Calculate PDF according to equation 6.4
(4) Find local minimums in density function
(5) **for each** mode a' in a
(6) $\Sigma' = \Sigma' \cup a'$
(7) **end for each**
(8) Calculate needed statistic parameters (mean μ and standard deviation σ)
(9) **end for each**

$$f(t) = \frac{1}{N} \sum_{i=1}^{N} k(\mathbf{t}_i; t) \tag{6.1}$$

where $N \in \mathbb{N}$ is the number of time values in the vector of observations and $k(\mathbf{t}_i; t)$ is a non negative kernel function

$$\int_{-\infty}^{\infty} k(\mathbf{t}; t)dt = 1. \tag{6.2}$$

As underlying probability distribution, we use the Gaussian distribution, which is defined as:

$$G(\mu, \sigma^2, t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{6.3}$$

where σ^2 is the bandwidth (smoothing factor), μ the mean value and t is the time value, for which the probability is calculated.

The choice of the bandwidth is important for the correctness of the results and it is the subject of research in different publications (e.g. [BGK10]). In the case of identifying the normal behavior of production plants, it is useful not to use a fixed value for smoothing factor but to keep it variable. Here, the variable smoothing factor is 5% of the current value. This results in the greater variance for greater time values and smaller variance for smaller time values. Therefore, the density is estimated as:

$$f(t) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\sqrt{2\pi} \cdot 0.05\mathbf{t}_i} e^{-\frac{(x-t)^2}{2 \cdot 0.05\mathbf{t}_i}}. \tag{6.4}$$

In the next step (line 4 in Algorithm 2) the local minimums in the calculated PDF are localized. One mode is assumed to be between the local minimums. Each detected mode is then included in the extended set of symbols Σ' (lines 5-7). At this point, the algorithm does not distinguish between multi-mode and single-mode PDFs.

If there exists only one mode in a PDF, the corresponding event is also included in Σ'.

Finally, referring to the original data (discrete time values) and based on the assumption of normally distributed data, the needed statistic parameters (mean μ and standard deviation σ) are calculated (line 8). This is done for each mode: between the minimum value, all local minimums and the maximum value.

Example 7. *The algorithm is illustrated using an artificial data set.*

Figure 6.11 shows an artificial data set with a vector of observations (discrete time values t for an event e). For the experiment four modes with the mean values [100; 200; 300; 400] and the corresponding standard deviations [0.1; 20; 10; 5] are used. For each mode 1500 samples were created randomly according to the given parameters.

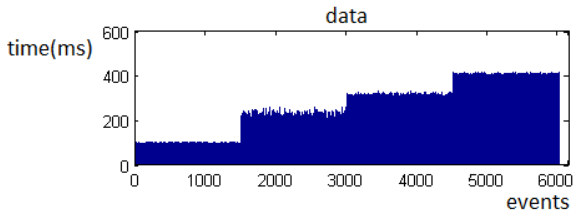


Fig. 6.11 Artificial data set for the running example.

Figure 6.12 shows the calculated probability density function. For the reason of comparison with other density functions, the function is normalized to the maximum value of '1'.

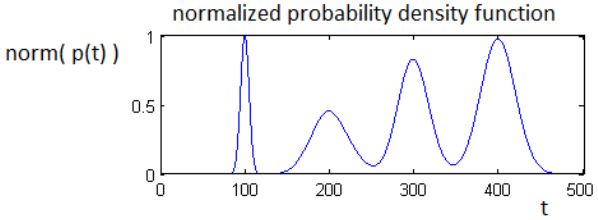


Fig. 6.12 Normalized probability density function.

As can be seen in Figure 6.13, all four distributions are localized. The estimated parameters are [100.013, 199.294, 299.481, 400.079] for the mean values and [0.541, 19.59, 10.687, 4.944] for the corresponding standard deviations.

Table 6.2 shows the given parameters μ and σ and the corresponding identified values. As long as the distribution functions are not mixed with each other, this method is able to distinguish between different modes that belong to different process steps.

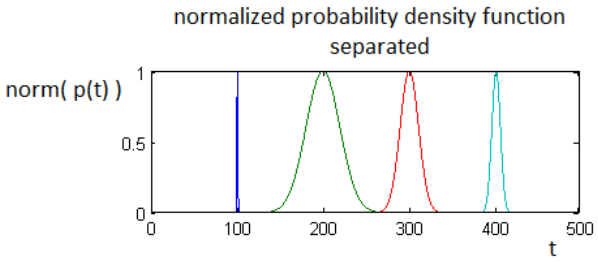


Fig. 6.13 Normalized probability density function, separated into the modes.

Table 6.2 Experimental results on artificial data.

Given μ	Identified μ	Given σ	Identified σ
100	100.013	0.5	0.541
200	199.294	20	19.59
300	299.481	10	10.687
400	400.079	5	4.944

In this example, all modes could be detected correctly. However, errors occur in some cases, especially with a high amount of overlapping values between modes. The separation capabilities and the error are theoretically analyzed in Section 7.2.2.

The result of this algorithm is an extended set of symbols, where new symbols are created for each event with a multiple-mode PDF according to the number of modes. Events with single-mode PDFs are not changed or modified. In the following steps (generation of the prefix tree acceptor and state merging), only the new set of symbols Σ' is used. Events with equal signal changes but stemming from different distributions are treated as different.

6.2.4 Step 2: Construction of the Prefix Tree Acceptor

In this section, we introduce two versions for the construction of a timed prefix tree acceptor. An offline and an online version. The offline version should be used preferably. However, if the network measurements can not be stored for some reason, the online version can be used.

6.2.4.1 Offline PTA creation

The modified alphabet Σ' with the identified events and the corresponding timing (created in Algorithm 2) is used to create the prefix tree acceptor.

Beginning with an initial state, each event creates a new state with a transition in between (including the event, the relative time value and an occurrence counter). The first learning example creates a linked list of states. In the following learning example, beginning with the initial state, it is checked whether it is possible to follow the current prefix tree acceptor. If at some point, there exists no appropriate transition,

a new transition to a new state will be created. Here, the timing probability density functions are considered already.

Example 8. *As can be seen in Figure 6.14, events with different timing probability density functions lead to different sub trees. Unlike the events 'b' and 'c', 'a' comprises two modes. When following the existing path, it is now not only checked whether the event is equal but also the equivalence of the corresponding PDFs. In this example, the occurrence of the event 'a' can lead to two different states. This avoids an unnecessary split operation afterwards.*

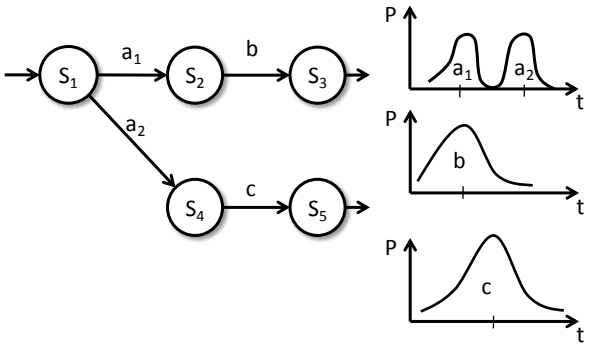


Fig. 6.14 Creating timed prefx tree acceptor using events with single/ multiple modes.

Algorithm 3 shows the procedure of generating a timed prefix tree acceptor.

The algorithm begins with the initial state (state 0) in line 1. The main part is organized in two for-loops. The outer loop is for the iteration over the learning examples D (line 2) while the inner loop is for the iteration over the events in each learning example (line 3). In line 4, the next event in the learning example is picked up, in line 5 the corresponding time stamp. The if condition in line 6 checks whether the current event can be assigned to some out-going transition of the current state.

If there exists such a transition, the timing in the transition is adapted (check for time ranges with minimum and maximum in line 6), the counters for the current state (line 8) and for the current transition (line 9) are increased and the destination of the taken transition is set to be the next current state (line 10).

If there exists no appropriate transition, first, a new state is created (line 12) and then a new transition between the current and the new state (line 13). Then, the new state is set to be the new current state (line 14) and the counter is initialized to 1.

After leaving the first for-loop (the last event of a learning example), the final count of last current state is increased (line 18).

Finally, after the iterations over all learning examples are finished, the created PTA is returned (line 20).

Note, that the timing constraints do not have to be calculated for the transitions, since each event is already assigned with the corresponding (separated) distribution function. However, the time ranges are adapted in line 7. In this function, an *if*-condition checks whether the time stamp δ is below the minimum or above the maximum value of the time range, which is adapted if necessary.

Algorithm 3 Generation of a timed prefix tree acceptor.

Algorithm timedPTA (Σ', D):
Given:
(1) extended set of symbols Σ' (output from Algorithm 2)
(2) Observations $D = \{\mathbf{D}_1, \ldots, \mathbf{D}_n\}$ where $\mathbf{D}_i \in (\Sigma' \times \mathbf{R})^*$,
$\mathbf{D}_i$ is one sequence of timed events (see Definition 2)
Result: timed Prefix Tree Acceptor $\mathcal{PTA}$ ($\mathcal{PTA}$ is an automaton according to Definition 15)

```
(1)   currentState = 0
(2)   for i=1 : n // iterate over sequences D_i
(3)     for j=1 : size(D_i) // iterate over events
(4)       e=getEvent() // e ∈ Σ'
(5)       δ=getTimeStamp()
(6)       if out-goingTransitionExists(State currentState, Event e)
(7)         adaptTiming(currentTransition, δ)
(8)         currentState.num++
(9)         currentTransition.num++
(10)        currentState = destination(State currentState, Event e)
(11)      else
(12)        createNewState(State newState)
(13)        createNewTransition(State currentState, Event e, Time δ, int num=0, State newState)
(14)        currentState = newState
(15)        currentState.num = 1
(16)      end if
(17)    end for
(18)    currentState.finalCount++
(19)  end for
(20)  return PTA
```

Usually, the learning examples D are separated according to the cycles in the observed production plants. This is part of the necessary expert knowledge.

A reasonable question is about the automatic identification of production cycles, which are used to begin with the initial state again. This could be done by defining start and stop conditions, e.g. pressing a start and stop button.

However, it is not necessary to identify cycles. A prefix tree acceptor in the special form as linked list of states including several cycles is sufficient to identify these cycles during the learning process. However, including this information increases the learning speed since many compatible states don't have to be identified and merged as the prefix tree acceptor already comprises these states.

6.2.4.2 Online PTA creation without preprocessing

So far, the timing behavior is evaluated before creating the prefix tree. If an event shows a multi-modal timing behavior, the event is split based on the identified timing constraints. In the subsequent generation of the prefix tree, this new set of events is taken into account. However, methods which find the separation points are only usable in a preprocessing step. For the case in which a timing preprocessing is not feasible (for instance when the observations can not be stored), an online generation of the PTA is necessary. Here, two other methods are introduced: *(1)* Online PTA

creation with local timing update and *(2)* Online PTA creation with global timing update.

To show the differences between both versions, they are both illustrated using an example with the same data set.

1. **Local timing update:**
   During the creation of the prefix tree acceptor, the timing values are successively added in each transition. These timing values are only stored locally in the histogram of the current transition after each incoming event.

   Example 9. *Let us consider the following three data samples (each consisting of events and corresponding time stamps): 1: (a7; b3; c2; d5), 2: (a5; b2; c1; a6), 3: (a3; c3; d5). Figure 6.15 illustrates the generation of the prefix tree acceptor. The first learning example leads to a linked list of states. In the subsequent steps, the event with the corresponding timing is captured locally in each transition.*

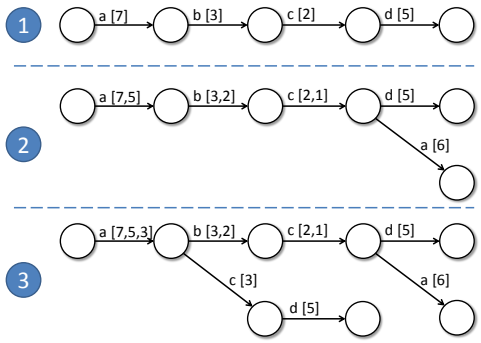


Fig. 6.15 Creating the timed PTA with local timing updates.

   In contrast to the method with time preprocessing, equal events occurring in different states are not mixed up. On the other hand, collecting time values separately leads to a small number of timing values per event, especially at the leafs of the PTA. This version is unusable for few data, since it is not possible to calculate statistics or probability density distribution functions for a small number of observations.
2. **Global timing update:**
   The second version is extending the first one. In contrast to the first version, the timing for each event is collected globally, i.e. the same timing histogram is stored in each transition with the same symbol $a \in \Sigma$. During creation of the prefix tree acceptor, all timing values in the prefix tree acceptor are updated.

   Example 10. *Let us again consider the following three data samples (each consisting of events and corresponding time stamps): 1: (a7; b3; c2; d5), 2: (a5; b2; c1; a6), 3: (a3; c3; d5). Figure 6.16 illustrates the generation of the prefix tree acceptor. It can be seen that equal events occurring in different paths in the PTA obtain the same timing histogram.*

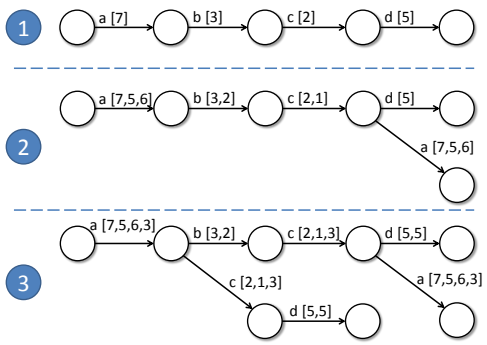


Fig. 6.16 Creating the timed PTA with global timing updates.

Using both versions, the separation of events based on computed timing constraints, e.g. using the kernel density estimation, can not be used because the postfixes are already mixed into common states. In that case, more complex splitting operations are necessary, e.g. as in [Ver10].

Comparing offline and online PTA generation, we can conclude the following:

Using the offline version, data with only few observations per event can be captured better, since all time values for equal events even from different states are collected. This leads to a higher amount of observations to calculate the density functions and the corresponding parameters. Furthermore, the preprocessing allows to separate events based on the timing (multi-modal PDF). This makes the splitting operation superfluous.

On the contrary, the online creation of the PTA leads to a smaller amount of data, especially in the leaves. Additionally, the time values are included on-line such that a separation of events based on the timing (multi-modal PDF) is not possible. In this case, a more complicated splitting operation is necessary (as introduced in [Ver10]). The online PTA creation should only be used if it is not possible at all to store the observations to use them in a preprocessing.

6.2.5 Step 3: State Merging

The PTA itself is not suitable for anomaly detection, since it is not generalized enough.

Once the PTA is created, it is used to identify compatible states that can be merged. After finishing this procedure, the identified automaton represents the timing behavior of a system in a generalized way. The resulting automaton is suited for the anomaly detection, which is the purpose of the identified automata.

This subsection describes the core of the identification algorithm: the state merging.

BUTLA uses a function *compatible* to check whether two states can be merged. The idea is similar to ALERGIA's approach (see section 3), with the difference that we compare incoming rather than outgoing transitions.

First of all, several additional variables are needed in Algorithm 4: The number of occurrences of an incoming transition for a specific state and a specific event ($f(a, \delta, v)$, line 1), the number of occurrences of incoming and outgoing transitions for a specific state ($f_{in/out}$, lines 2-3) and the number of measurement sequences which end in a specific state (f_{end}, line 4). If the f_{end}s for two states (in relation to f_{in}) are too different, they are not merged (lines 5-6). Similarly, if for any event a the corresponding $f(a, *)$s are too different (in relation to f_{in}), the states are also not merged (lines 8-9).

If the two nodes are found to be compatible, the compatibility of the respective subtrees must be checked, too. This is done by applying the algorithm 4 recursively to all nodes in the subtrees (lines 10-11).

Algorithm 4 Comparison algorithm *compatible*

Algorithm compatible (v,w):
Given: $v, w \in S$
Result: decision yes or no
(1) $f(a, \delta, v) := \sum_{e=(*,a,\delta,v)\in T} Num(e), v \in S, a \in \Sigma, \delta \in \Delta$, where $*$ is an arbitrary element
(2) $f_{in}(w) := \sum_{e=(*,*,*,w)\in T} Num(e), w \in S$
(3) $f_{out}(v) := \sum_{e=(v,*,*,*)\in T} Num(e), v \in S$
(4) $f_{end}(v) := f_{in}(v) - f_{out}(v), v \in S$
(5) **if** fractions-different$(f_{in}(v), f_{end}(v), f_{in}(w), f_{end}(w))$
(6) **return false**
(7) **for all** $a \in \Sigma$ **do**
(8) **if** fractions-different$(f_{in}(v), f(a, \delta, v), f_{in}(w), f(a, \delta, w))$
(9) **return false**
(10) **if** not compatible$(v', w')\ \forall\ (v, a, \delta, v'), (w, a, \delta, w') \in T$
(11) **return false**
(12) **end for**
(13) **return true**

This is done to prevent later unnecessary splits; the function *PDF-different* can be implemented using the well-known R^2 test.

To compare whether two fractions $\frac{f_0}{n_0}$ and $\frac{f_1}{n_1}$ are significantly different (function *fractions-different*), we use the Hoeffding Bound [Hoe63]:

$$\text{fractions-different}(n_0, f_0, n_1, f_1) := \left|\frac{f_0}{n_0} - \frac{f_1}{n_1}\right| > \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n_o}} + \frac{1}{\sqrt{n_1}}\right)$$

where $1 - \alpha, \alpha \in \mathbf{R}$ is the probability of the decision.

Example 11. *The computation of the compatibility of states is illustrated using an example with a simplified computation of the compatibility. Simplified, because the example only considers the compatibility computation of the states according to line 8 in Algorithm 4. However, the computation of the function fractions-different for final states (line 5) works in the same way, as well as the recursive compatibility check (line 10).*

Figure 6.17 shows two states with incoming and outgoing transitions for which the compatibility is computed.

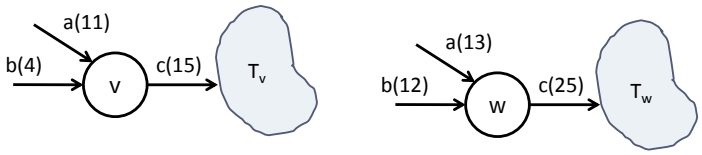


Fig. 6.17 Example states for the computation of the compatibility.

Table 6.3 shows the computation of the compatibility for two configurations. The numbers of configuration 1 are taken from Figure 6.17. In configuration 2, the occurrence counter for the symbol a in state w was increased by one. For the computation, we used α=0.8.

Table 6.3 Example computation of state compatibility.

		configuration 1 (Figure 6.17)	configuration 2 (slightly modified)
all symbols	$f_{in}(v)$	15	15
	$f_{in}(w)$	25	26
symbol a	$f(a,\delta,v)$	11	11
	$f(a,\delta,w)$	13	12
	fractions-different	true	false
symbol b	$f(b,\delta,v)$	4	4
	$f(b,\delta,w)$	12	12
	fractions-different	true	false
result	compatible	false	true

The computation of the function fractions-different for configuration 1 gives the result that the states are compatible. However, it can be seen that only slight modifications can decide on the compatibility of states. By changing one value (configuration 2), the states are not considered to be compatible any more.

The compatibility computation of two states is prone to errors. Especially when only few observations are available, a small change (e.g. the addition of only one observation) can decide on the compatibility of two states, leading to a different automaton. This effect can be seen in Example 11.

Additionally, there exists the parameter α which has to be set manually by the operator. Although there exist typical values which can be set, it is still a kind of expert knowledge and a wrong setting of this parameter can lead to either a over-approximated automaton (too many merges) or to an under-approximated automaton (too few merges). For instance, both states from Figure 6.17 are considered not to be compatible if the α-parameter is set to α=0.7 in the example. A detailed theoretical analysis on the identification error follows in Section 7.2.

6.3 Online Timed Automaton Learning Algorithm (OTALA)

The implementation of identification algorithms on cyber-physical systems such as IO devices in industrial networks puts several additional requirements to the algorithms, which are mainly:

- **Economical usage of memory space:** The implementation of identification algorithms on cyber-physical systems requires an economical use of memory space. Due to the limited memory space, the observations can not be stored (as it is required by offline learning algorithms) but rather every data sample has to be included directly into the model. This leads to the need of online learning algorithms.
- **Real-time capability:** The real-time capability is an important requirement for the learning in cyber-physical systems. Since the incoming data can not be stored, the implementation has to be able to handle the data stream in real-time, i.e. each data sample has to be included into the model before the next data sample arrives.
- **No expert knowledge:** Furthermore, it is required that as little as possible expert knowledge is used. Offline learning algorithms still need some expert knowledge: the learning cycles have to be recorded. Here, the user of the learning algorithm usually has no knowledge about the amount of data that is needed to learn a correct automaton. The main advantage of our proposed algorithm is that it copes without expert knowledge. Additionally, the implementation can be operated by untrained personnel: Using a convergence criterion, the algorithm recognizes itself when the identification process is finished.
- **Active learning is not possible:** Active learning algorithms can not be used for learning in Cyber-physical production systems since there is no possibility to ask for samples. The data has to be taken as it comes. Additionally, there is no oracle (teacher) which can tell whether the hypothesis is correct and eventually return a counter example.

To overcome these challenges, this section introduces an unsupervised online learning algorithm, the *Online Timed Automaton Learning Algorithm (OTALA)*.

The key question for automata identification algorithms is about the compatibility of states. Offline algorithms following the state merging approach (BUTLA and other) use the event sequences of the postfixes to determine the compatibility of states. However, learning the automaton in an online manner, it has to be referred to other information.

OTALA uses the signal vector **u** as state information.

Definition 26 (Signal Vector). A signal vector **u** is a vector of the input and output signals in the observed system $\mathbf{u} = (io_1, io_2, ..., io_{|IO|})^T$, where each io_i is the value of a discrete signal (input or output) in the observed system and $|IO|$ is the number of input and output signals. Each $io_i, i \in \mathbb{N}$ indicates for the corresponding input or output signal, whether it is active ($io_i = 1$) or inactive ($io_i = 0$).

Therefore, the identification algorithm OTALA is mainly based on the following assumption:

Assumption 1 *Each state in the observed cyber-physical production system can be represented by a signal vector and each signal vector corresponds to one state in the final automaton.*

This assumption is necessary, since the compatibility of states has to be determined during runtime and, especially at the beginning, no information about event sequences is available. This method implies that the system is memory-less, i.e. a state holds no information about the history, the prefix. Different event sequences can lead to the same state. A similar formalism is used by [RLL10].

The formalism of Timed Automaton (see Definition 13) is modified to additionally capture the signal vector as state information.

Definition 27 (State-based Timed Automaton). The State-based Timed Automaton is a 4-tuple $A = (S, \Sigma, T, \delta)$, where

- S is a finite set of states. Each state $s \in S$ is a tuple $s = (id, \mathbf{u})$, where id is a current numbering and $\mathbf{u} = (io_1, io_2, ..., io_{|IO|})^T$ is a signal vector according to Definition 26.
- Σ is the alphabet, the set of events.
- T is a set of transitions. A transition is represented with (s, a, δ, s'), where $s, s' \in S$ are the source and destination states, $a \in \Sigma$ is the symbol and δ is the clock constraint.The automaton changes from state s_i to state s_j triggered by a symbol $a \in \Sigma$ if the current clock value satisfies δ. The clock c is set to 0 after executing a transition, so that the clock starts counting time from executing this transition.
- A transition timing constraint $\delta : T \rightarrow I$, where I is a set of intervals. δ always refers to the time spent since the last event occurred. It is expressed as a time range or as a probability density function (PDF), i.e. as probability over time.

The State-based Timed Automaton does not need an initial state as the original Timed Automaton does. Since the state is defined over the active/inactive IO values, the starting point is the state corresponding to the actual system's configuration. Furthermore, due to the infinite operation of the system, the set of final states is not needed either. Finally, (just like the PDTA from Definition 15) the State-based Timed Automaton does not need multiple clocks. Instead, only one clock is used which is reset with each firing of a transition, which results in relative time since the entering of states.

The algorithm OTALA works as follows (see also algorithm 5):

First of all, an empty automaton $\mathcal{A}$ is created according to Definition 27 (line 0). In lines 1 - 2, the starting state is created using the first signal vector which is obtained by the function *getNextEvent(D)*. Since it is an online identification procedure, the data come from a running system and eventually it has to be waited until a new event occurs. This is done in the function *getNextEvent(D)*. It additionally returns the corresponding time stamp, which, however, is not used in line 1.

The main part of the algorithm is wrapped by the while-loop (lines 3 - 22), which is executed as long as the identification has not converged. This is checked in the function *identificationConverged()*. How the convergence of the identification is detected will be explained later.

In line 4, the function *getNextEvent(D)* is used again to obtain the next signal vector $\mathbf{u}$ and the corresponding t.

The following for-loop (lines 5 - 15) is used to search for the state that corresponds to the obtained signal vector. In the iteration over all states of the automaton, line 6 checks whether the obtained signal vector corresponds to the signal vector of the state. The signal vector of the state is returned by the function *getSignalVectorFromState(s)*, which gets the state number as input. If such a state already exists in the automaton

(line 7), it is checked whether there exists a transition between the current state and the new state (line 8).

If there exists such a transition, the timing information for this transition is adapted (function *adaptTimingInformation(t)*, line 9), otherwise a new transition is created (function *createNewTransition(currentState,t,s)*, line 11) with the current state as source, the new state as destination and the time stamp t setting both, minimum and maximum. Finally, the new state is set to be the next current state (line 14).

If no state with the current signal vector exists (line 16), a new state is created (function *createNewState()* line 17), then a new transition between the currents and the new state is created (function *createNewTransition(*$currentState, t(j), s_new$*)*, line 18) and finally the new state is set to be the next current state (line19).

After each step (iteration over all states and modification of the automaton), it is checked whether the identification process converged (function *identificationConverged()*, line 21). If the identification process converged, the identified automaton $\mathcal{A}$ is returned (line 22), otherwise the while-loop is continued in line three, waiting for the next event.

Algorithm 5 Online timed automata learning algorithm OTALA.

Algorithm OTALA:
Given: learning examples $D = \{D_1, D_2, ..., D_n\}$ according to Definition 2
Result: State-based Timed Automaton $\mathcal{A}$ according to Definition 27

```
(0)   A = initializeAutomaton() // A is a State-based Timed Automaton A according to Def. 27
(1)   t, u = getNextEvent(D)
(2)   currentState = createNewState(u)
(3)   while identificationConverged()==false
(4)     t, u = getNextEvent(D)
(5)     for all s ∈ S, where S is the list of states in the current automaton A
(6)       if u == getSignalVectorFromState(s)
(7)         stateExists = true
(8)         if transitionExists(currentState, s, A)
(9)           adaptTimingInformation(t)
(10)        else
(11)          createNewTransition(currentState, t, s)
(12)        end if
(13)      end if
(14)      currentState = s
(15)    end for
(16)    if !stateExists
(17)      s_new = createNewState()
(18)      createNewTransition(currentState, t(j), s_new)
(19)      currentState = s_new
(20)    end if
(21)    if identificationConverged()=true
(22)      return A
(23)    end if
(24)  end while
```

Figure 6.18 illustrates the principle of the learning algorithm OTALA: The signal vectors are processed subsequently. The first signal vector creates the first state. For

the following signal vectors, it is checked whether a state with the corresponding signal vector and the corresponding transition already exists and if not, a new one is created. This is continued until no more events are observed or the learning converged.

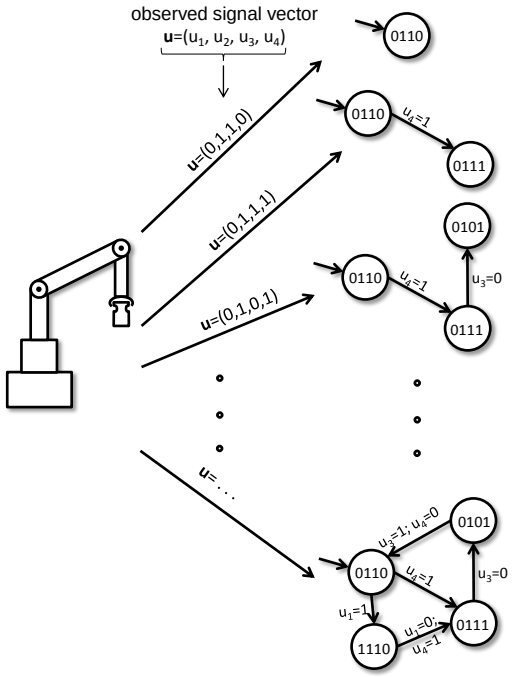


Fig. 6.18 The principle of the online learning algorithm OTALA.

The reader may has noticed that the runtime is exponential: In each step (iteration over all states and modification of the automaton), all possible states have to be checked for equivalence. The number of possible states is given by the number of possible combinations of each input/ output signal: $|S| = 2^{|IO|}$, where $|IO|$ is the number of signals, i.e. it is exponential in the number of input and output signals. However, in practice, the number of states is mostly rather small, so that this approach still can be applicable. Furthermore, the identification runtime depends on the number of transitions in the behavior model. The maximum number of outgoing transitions for one state corresponds to the number of states since between two states there can be only one transition.

The runtime behavior will be analyzed in detail in Section 7.6. Additionally, the runtime behavior will be compared with the runtime of the offline identification with BUTLA.

An additional feature of the OTALA algorithm is that it autonomously recognizes when the learning process is finished.

The learning progress can be measured based on the following characteristics:

1. **Number of states:** During the learning process the number of states is continuously growing. Whenever a new state is observed, which is not available in the model, it is added to the model.
2. **Number of transitions:** Additionally, the number of transitions has to be considered. Even when no states are added anymore, transitions between states can still be added.
3. **Changing (enlarging) the time bounds:** Finally, the changing of the time bounds has to be considered. Even when no states and transitions are added to the model anymore, the time bounds (the minimum and maximum time value for each transition) can still change

The identification can be considered as finished, when no more states and transitions are added to the model and when the time bounds of the transitions do not change anymore. In practice, this is not easy to decide, since we cannot look into the future and therefore we cannot know if one of the values will change in the future.

So we have to decide based on the past that whether learning is finished or not. If for a *certain amount of time* nothing has changed, the learning process can be considered as finished. The question here is: What does "certain amount of time" mean?

Figure 6.19 shows the convergence of the learning for a test data set.

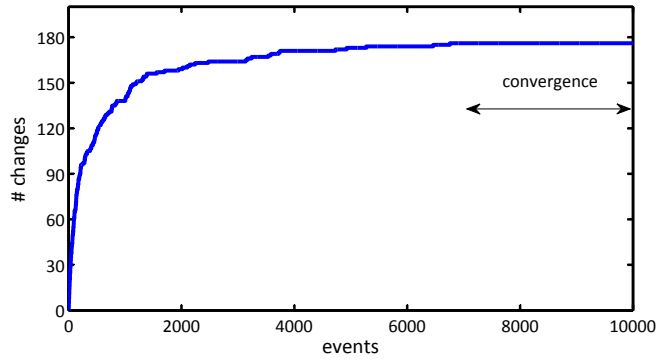


Fig. 6.19 Convergence of the learning algorithm OTALA

A trivial yet efficient possibility is to consider the last n_{conv} samples and check whether the model has been changed concerning the characteristics in the aforementioned enumeration. The value n_{conv} can be chosen arbitrarily, a good choice is dependent on the observed process. However, setting $n_{conv} = 1000$ is empirically a good choice in most cases.

Another method takes the variable number of states into account. For this, the number of states is multiplied with a factor f_{conv}, which also can be chosen arbitrarily. Empirically, $f_{conv} \approx 10$ in most cases is a good choice. Hence, the identification converged if the last $n_{conv} = |S| \cdot f_{conv}$ events did not contain any changes ($|S|$ is the number of states in the automaton).

6.4 Anomaly Detection

The identified models are finally used for anomaly detection. From a practical point of view, the anomaly detection is the comparison of the behavior of the real plant with the prediction of the identified model. An anomaly is a deviation of the simulated behavior from the current behavior of the real plant. In the literature sometimes the term *novelty detection* is used (e.g. [MS03b, MS03a]), i.e. an anomaly is signaled if some new behavior is observed which has not been encountered before.

From a theoretical point of view, the issues have to be investigated in more detail. Here, still several open issues remain:

- **Simulation inaccuracies:** The models which are used for anomaly detection have been identified automatically beforehand using an identification algorithm. The goal of identification is to abstract a generalized behavior model from the measurements, i.e. the model is an abstracted view of the original observations. To obtain an abstracted view, a generalization of the observed behavior is necessary. This leads to a normal imprecision. During anomaly detection, this imprecision has to be taken into consideration.
- **Handling of time:** Cyber-physical production plants usually depend on time. Therefore, pattern based methods that compare observed and identified behavior patterns are only suitable to a limited extent. Using the timing information for anomaly detection, effects such as a time offset or different time velocities have to be considered.
- **Noise:** All technical data such as sensor values are subject to noise. So, for any discrepancy between predictions and measurements it must be decided whether it has been caused by noise or by an anomaly.
- **Trade-off between false-positive and false-negative rate:** This issue is closely related to *simulation inaccuracies*. Error sources such as noise and simulation inaccuracies lead to classification errors. Two measurements are important to assess the quality of an anomaly detection algorithm: The false-positive rate (behavior that is free of anomalies but recognized as anomalous) and the false-negative rate (anomalous behavior which is not detected as such). A high false-positive rate leads to operator attention overload, while a high false-negative rate causes system degradation due to the anomalous system behavior. Therefore, a good anomaly detection is a trade-off between the false-positive and false-negative rate.

This section introduces the anomaly detection algorithm ANODA. For this, the identified PDTA as defined in Definition 15 is used.

The anomaly detection algorithm was firstly introduced in [MNV+11], while the abbreviation "ANODA"' was firstly used in [VBNM11].

The behavior of the production plant can be defined as a path through the automaton (see also Figure 6.20):

Definition 28 (Path through Automaton). Let $A = (S, s_0, F, P, \Sigma, E)$ be a PDTA according to Definition 15. A path P through the automaton is defined as a sequence of transitions, i.e. $P \subseteq T$.

Furthermore, the observations in the plant are defined as follows:

Definition 29 (Observation). One observation in the plant is defined as $o = (a, t)$, where

- a is the trigger event (signal change) in the plant according to Definition 3 and
- $t \in \mathbb{R}$ is a relative time value (relative to the last event).

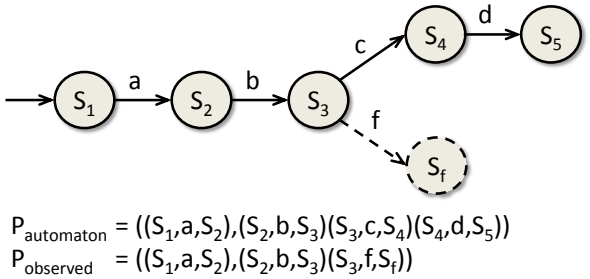


Fig. 6.20 An identified path in the automaton ($P_{automaton}$) and a currently observed path through the automaton ($P_{observed}$) with an anomaly at event f.

The identified automaton is now used to detect an unusual behavior (an anomaly) in the behavior of a cyber-physical production system. During runtime, the running production plant is observed and the identified model is simulated in parallel. Then the simulation outputs are compared with the observations from the running system. If any difference arises there, an anomaly has occurred.

Algorithm 6 shows the discrete anomaly detection algorithm ANODA. An identified PDTA (according to Definition 15) is given as input. Furthermore, observations from the production plant are received regularly.

The result is the detected anomaly if there exist one, otherwise the target state s_{new} of the taken transition is returned. This state is used as input for the next iteration of the ANODA algorithm.

The discrete anomaly detection in Algorithm 6 works as follows:

In line 1 it is checked, whether there exists a transition from the current state s (which is given as input) to some destination state s' using the given event a. If no such transition exists, the anomaly "unknown event" is returned in line 8.

The timing is checked in a separate if-condition, such that it can be distinguished between the anomalies "unknown event" and "wrong timing". For this, in line 2 it is checked whether the given time stamp t satisfies the timing constraint δ of the chosen transition from line 1. Here, time ranges can be used as well as probability density distribution functions. In the first case it has to be checked, whether the provided time stamp t lies between the identified minimum and maximum time value (see also Figure 6.4). In the latter case, it has to be checked whether the provided time stamp t satisfies the identified PDF by calculating the probability of occurrence for this time stamp. An anomaly is signaled when the calculated probability is below a threshold (this threshold is an additional parameter). This case is illustrated in Figure 6.5. If the time stamp does not satisfy the given timing constraint, in line 5 the anomaly "wrong timing" is returned.

If both, the event check in line 1 and the timing check in line 2 give positive results, the chosen destination state from line 1 is set to be the next current state in line 3. It is returned in line 10.

Algorithm 6 Discrete anomaly detection algorithm ANODA

Algorithm discreteANODA:
Given:
(1) Probabilistic Deterministic Timed Automaton (PDTA) $A = (S, s_0, F, P, \Sigma, T)$ (according to definition 15)
(2) An observation $o = (a, t)$ according to Definition 29
(3) $s \in S$, at the beginning, this value is assigned with the initial state s_0

Result: detected anomaly (if there exists one), otherwise the moved current state s_{new}

```
(1)   if exists e ∈ T with e = (s, a, ∗, s′) then // a ∈ Σ is a symbol,
                                                  // s′ ∈ S is a potential destination state,
                                                  // and * is an arbitrary time stamp (not checked here)
(2)     if t satisfies δ(e) then // check timing
(3)       s_new := s′ // go to next state
(4)     else
(5)       return anomaly: wrong timing
(6)     end if
(7)   else
(8)     return anomaly: unknown event
(9)   end if
(10)  return s_new
```

Following types of anomalies can be detected using this procedure:

- **Unknown event/ Wrong event sequence:** In the current state, an event occurred that has not been observed before (in the identification process). E.g. while filling a bottle, the next event should be "bottle full" (identified event), but for some reason the filling stops (current event: "stop filling").
  For every observed event, it is checked whether its symbol corresponds to one of the possible outgoing events in the current state (line 1). An error is found when no transition with the observed event exists, i.e. if the observed path is not equal to one possible simulated path in the automaton (see also figure 6.20).
- **Timing error:** A timing error occurs when a signal changes correctly, but the observed timing (relative time stamp) does not satisfy the identified timing constraint. E.g. if the filling of the bottle should take between four and five seconds, an anomaly would be found when this takes less than four or more than five seconds. Since we often do not have hard time limits, it's useful to work with distribution functions. In this case the probability of the failure can be returned.

Additionally, the following two errors can be detected based on the information available in the automaton, though it is not covered by Algorithm 6.

- **State remaining error:** When the system remains in a state which is not a final state, it is assigned to be an error. E.g. for a production cycle with six bottles to fill, it is an error when the production stops at the fourth bottle.

To detect this kind of anomalies, an "active waiting" for the next event has to be implemented. While waiting for the next event, it has to be checked continuously whether it is still possible to use one of the out-going transitions. For this, the timing constraints for all outgoing transitions are monitored. If the upper limit is crossed for all out-going transitions, such that (if in the future an event will occur) the timing constraint cannot be satisfied, it is checked whether the current state is a final state ($s \in F$).
A state remaining error has occurred when the upper limits of the timing constraints of all outgoing transitions are crossed and $s \notin F$.

- **Probability error:** The PDTA, according to Definition 15, holds information about the probabilities of all outgoing transitions. Taking these probabilities into consideration, more complex and gradual errors can be detected when the probabilities in the observed system diverge from the probabilities in the model. E.g. while checking the filling of a bottle, 95% of the bottles are filled correctly and 5% incorrectly. Here it would be an anomaly, if the observed probability exceeds this usual value.
To detect this kind of anomalies, the occurrences of each event in the observed system are counted and the probabilities are recalculated after each occurrence. Using an additional parameter $\psi \in [0, 1]$, which is needed as permitted tolerance, an anomaly is signaled if the probability exceeds this tolerance.

In Section 7.2, the anomaly detection capabilities and the error rates for wrong event sequences and timing errors are evaluated.

The anomaly detection algorithm is explained using an automaton according to Definition 15. Indeed, the algorithm does not distinguish between automata which are identified using BUTLA or OTALA. Both algorithms operate on nearly the same data structure. The only difference is that OTALA additionally uses the signal vector in the states, which is needed during the identification process only. However, OTALA includes the events into the transitions in the same way as BUTLA does. These events are used to be compared with the observations in the production plants (see Definition 29). Therefore, automata from both identification algorithms can use ANODA for anomaly detection.

6.5 Adaptive Learning

In Section 6.2, we have seen that BUTLA is not able to recognize the convergence of identification autonomously. Therefore, it often occurs that a model is not identified completely. This leads to the fact that often correct behavior is classified as faulty, i.e. we obtain a high false positive rate. OTALA is, however, capable to recognize the convergence of identification autonomously. Nevertheless, the same problem can still occur.

A possible solution is to adapt the model during runtime: If the anomaly detection algorithm signals a faulty behavior, the plant operator can decide whether it is really due to an error or whether it is a false alarm. In the latter case, the observed behavior has to be included into the model.

In this section, we show how this new behavior can be included into the model afterwards. However, the adaptive learning is still subject of future work. Here, we only sketch the rough methodology.

6.5.1 Adaptive Learning for BUTLA

Using the models identified by BUTLA, it is not a trivial case to adapt the model, since BUTLA determines the compatibility of two states based on their postfixes. However, the postfix of the current behavior is not yet known at the time of failure.

It can be distinguished between two possible failure types:

1. *Timing error:* This failure type indeed is easy to adapt in the model: The given time bounds just have to be enlarged by the supposed wrong timing value and the probability density function with the corresponding parameters have to be calculated again. The structure of the automaton (states and transitions) does not have to be changed.
2. *Wrong event sequence:* This case is more difficult to handle. At this stage we only give a workaround for the adaption: Since the compatibility of two states is not encoded in the state itself (as for example the IO vector used by OTALA) but recursively calculated over event sequences of the postfixes of both states, we first have to observe the complete production cycle until the end. This requires saving the observations (event sequence) of the whole production cycle. These observations are included into the prefix tree acceptor (which by the way also has to be stored beforehand). Then BUTLA is applied to identify a new automaton based on the modified prefix tree acceptor.

6.5.2 Adaptive Learning for OTALA

A model, which is identified with OTALA can be adapted easily, since OTALA uses the IO vector to determine the compatibility of two states. Adapting a misclassified timing error to the automaton is the same as for BUTLA: Enlarging the time bounds and compute the probability density function over time and its parameters again. In case of a misclassified event sequence error, it has to be checked whether the new observed state (IO vector) already exists. This is done in the same way as OTALA works:

1. If the corresponding state exists, we just have to include a new transition (with the time value setting the time bound) between the current state and the found target state.
2. If the corresponding state does not exist, we have to create it with the corresponding transition in between. The same is done for the subsequent events and IO vectors until we reach a state that is already available in the automaton (case 1).

Chapter 7

Theoretical Results

In this chapter, the introduced algorithms BUTLA (see Section 6.2) and OTALA (see Section 6.3) are analyzed according to some theoretical aspects.

The contributions of this chapter are the following:

- We show that BUTLA runs polynomial to the input size, i.e. that BUTLA identifies the class of 1-clock timed automata weakly in the limit with probability one (Lemma 1).
- We calculate the learning error for each mentioned error type (Section 7.2).
- We give some constraints under which we can show that BUTLA identifies the class of 1-clock timed automata strongly in the limit with probability one (Theorem 11).
- we subdivide the framework of identification in the limit into *states identification in the limit* (Theorems 10 and 11) and *timing identification in the limit* (Theorem 12).
- We show that the bottom-up strategy works faster than top-down in relevant cases (Proposition 6).
- We show that BUTLA identifies the timing behavior without a splitting operation, and that the computation time can be reduced using a preprocessing of time values instead of the splitting operation (Propositions 7 and 8).
- We show that despite the exponential runtime of OTALA (Lemma 2), it can still be used efficiently for the identification of timed automata.
- We evaluate the applicability of online and offline identification algorithms for certain application scenarios (Section 7.6).
- We show that the Anomaly Detection Problem using ANODA $P_{ANODA} \in \mathcal{L}$, i.e. the computation can be performed on logarithmic space (Theorem 13).

This chapter is organized as follows: Section 7.1 analyzes the runtime of the identification algorithms BUTLA and OTALA. In Section 7.2, the error of the identification algorithms is analyzed. These are the state merging error and the timing error. Section 7.3 analyzes the convergence of the identification with BUTLA concerning the concept of *identification in the limit*. In the Sections 7.4 - 7.6, the identification approaches are compared: Top-down vs. Bottom-Up (Section 7.4), Splitting vs. Non-Splitting (Section 7.5) and Online vs. Offline Identification (Section 7.6). Finally in Section 7.7, the runtime of the anomaly detection algorithm ANODA is analyzed.

The results in this chapter were partially published in [NSV+12] and [NVM+13].

7.1 Runtime Analysis of the Identification Algorithms

First of all, we analyze the runtime behavior of both identification algorithms, BUTLA and OTALA. Both algorithms show a different runtime behavior. BUTLA is an offline identification algorithm, which identifies the TA based on the PTA and therefore the runtime depends on the number of states in the PTA. OTALA is an online identification algorithm. At runtime there exists no PTA, the structure of the automaton is built piece by piece while checking the vector of input and output signals. Therefore, the runtime of OTALA is not based on some number of states in a PTA but on the number of input and output signals in the observed system.

7.1.1 Runtime Analysis of BUTLA

In Section 6.2.5, we presented the identification algorithm BUTLA (Algorithm 1). This algorithm consists of three main parts:

1. Computation of events (step *(1)*)
2. Timing preprocessing (step *(2)*)
3. Generation of the prefix tree acceptor (step *(3)*)
4. Identification of the automaton structure and state merging (Step *(4-10)*)

Lemma 1. *The algorithm BUTLA runs in $O(n_0^3)$ where n_0 denotes the number of states in the prefix tree acceptor.*

Proof. Step (1) of the algorithm runs in $O(n_0)$ since the number of events is less than the number of observations.

Step (2) runs linearly to the number of observations (see Proposition 8).

Step (3) also uses every observation once, i.e. it runs in $O(n_0)$.

In step (4-10) the algorithm compares a maximum of n_0^2 nodes. In each merging step for nodes v, w, both of their subtrees have to be accessed by the algorithm compatible and determinized. □

Hence, BUTLA runs polynomially to the size of the prefix tree acceptor. This lemma will later be used for the proof of *identification in the limit*.

In the paper where Carrasco introduces the Alergia algorithm [CO94], he states that Alergia has a linear runtime. However, the runtime is not analyzed there in a formal way, but instead it's done by giving an empirical measurement. The runtime measurement there is based on the total number of input samples. For the experiment, he used many input samples generated from the Reber grammar [Reb67] (see also Section 8.1.1.1). To measure the computation runtime, he added the runtime of creating the PTA with the runtime of merging the compatible states. But since the number of input samples is very high while the number of states in the PTA is rather small (even for 10000 input samples, there are only 41 states in the PTA), the time for creating the PTA is much higher than the time for merging the compatible states. This effect is reinforced by the fact that for an increasing input size, the number

of states in the PTA stays constant, and thereby the runtime of the state merging. Adding this constant time for merging to the linear time of creating the PTA, the total runtime seems to be linear. This means that the quadratic term of the state merging can be neglected. This again leads us to the fact that it makes a difference if cycles in the data measurements are separated or recorded in one long linked list. In the second case, the number of states corresponds to the number of events since there is no overlapping of the prefixes. This is illustrated in the following example:

Example 12. *We compare the runtime of learning the Reber grammar with the given numbers from the last paragraph with the same number of events, which lead to a PTA as a linked list of states. The number of events, the number of states and the runtime (in time units) is given for both scenarios in Table 7.1*

Table 7.1 Comparison of two scenarios and their computation runtimes.

	separated input samples	all events in one sample
number of events	10000	10000
number of states in the PTA	41	10000
runtime [time units]	$10000 + 41^2$ $= 11681$	$10000 + 10000^2$ $\approx 10^8$

It can be seen that separating the cycles (with 10000 events in total) leads to a highly condensed PTA with 41 states and therefore to a runtime of 11681 time units. The quadratic runtime of the state merging plays a marginal role. Whereas the PTA of the second scenario, where the cycles are not separated, includes 10000 *states (each event leads to a new state) and therefore leads to a computation time of about* 10^8 *time units.*

In Section 8.2. the runtime of BUTLA is analyzed empirically based on artificial and real data.

7.1.2 Runtime Analysis of OTALA

The algorithm OTALA was introduced in Chapter 6.3 and is given in Algorithm 5. The reader may has noticed that the runtime of OTALA is exponential: In each identification step, all possible states have to be checked for equivalence with the incoming observed data vector.

Proposition 5. *The potential number of possible states is exponential to the number of IOs (number of input and output signals).*

Proof. Applying the assumption that each state in the observed cyber-physical production system can be described by the signal vector of the inputs and outputs and corresponds to one state in the final automaton, the number of possible states is given by the number of possible combinations of all input and output signals: $|S| = 2^{|IO|}$.
□

In each step (lines 2-12 in Algorithm 5), all of the possible states have to be checked for equality with the given IO vector in the worst case. This leads to the worst case runtime of OTALA:

Lemma 2. *The worst case runtime of OTALA per step (which is the search of the destination state corresponding to the input data vector) is in* $\mathcal{O}(2^{|IO|})$*, where* IO *is the number of input and output signals.*

Proof. This follows directly from Proposition 5 and line (2) in algorithm 5 in which all states are checked for equality. □

However, the worst case runtime will rarely be exhausted while running the identification algorithm in cyber-physical production systems, even with maximum number of states. The average runtime can be calculated as:

$$T_{avg,theo} = \frac{1}{|IO|} \sum_{i=0}^{|IO|-1} 2^i = \frac{1}{|IO|} \left(2^{|IO|} - 1\right) \tag{7.1}$$

To obtain a model with the worst case number of states, we have used the drunken sailor simulation (a random walk) to generate a model such that each state in a discrete state space will be visited at some point in time. Figure 7.1 shows the runtime behavior of the model identification of the drunken sailor simulation with 10 signals. The total number of states is $|S| = 2^{10} = 1024$. The growing number of states is shown in Figure 7.2.

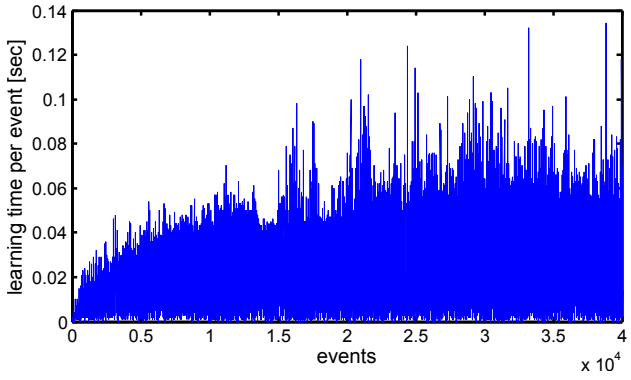


Fig. 7.1 Runtime behavior of OTALA.

In practice however, the systems show a better behavior. Often signals are only switched on and off together in a pair or group (e.g. two conveyor belts to transport something) or combinations of signal groupings can be precluded (e.g. the filling of a container can only be started if the container is not full). For each excluded combination possibility, the number of possible states reduces by half. This reduces the runtime of the identification process.

The runtime can further be reduced when it is possible to use more memory space. To achieve this, space for each possible state is allocated beforehand. This needs space exponential to the number of IOs. During the learning process, it is not necessary to iterate over the whole list of states. Instead of this, the IO vector just has to be converted into a decimal number (the time taken is logarithmic to the number of IOs) and used for a storage access. This approach is usable in practice, since the number of IOs is known beforehand, therefore the worst case memory space consumption can be determined.

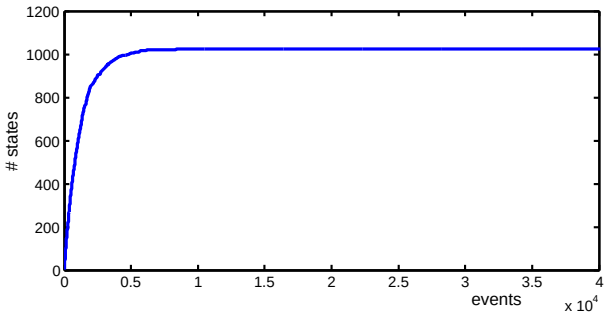


Fig. 7.2 State convergence behavior of OTALA.

Considering the Figures 7.1 and 7.2, it is notable that the number of states increases at the beginning and then remains constant. The runtime behaves in the same way: As long as there are only few states in the automaton, less time is needed. In reverse, once all states have been captured, the runtime remains relatively constant. This again leads to a linear cumulative runtime, considered over all incoming events.

The average runtime will be empirically analyzed in Section 8.2.2.

7.2 Evaluation of the Learning Error

In Section 6.2, the BUTLA algorithm was introduced, which learns a Timed Automaton from positive data. The identified models are used for anomaly detection. An appropriate algorithm, ANODA, has been introduced in Section 6.4. However, errors can occur during learning, which have an impact on the anomaly detection. In this section, the possible identification error is analyzed. The results in this section have been partially published in [NVM+13].

The identification error is divided into two parts: *(1)* the state merging error (Section 7.2.1) and *(2)* the timing identification error (Section 7.2.2). Both types of identification error are evaluated in this section.

In this section, for each aforementioned anomaly type, the probability is analyzed that the anomaly could not be found (false negative) or that an anomaly is identified incorrectly (false positive). Such errors can occur either because of incorrectly identified models or because of incorrectly classified measurements.

The calculation of the identification error will be used later to prove the *Identifiability in the limit* in Section 7.3.

7.2.1 State merging error

One main part of the identification algorithm BUTLA is the identification of the automaton structure, i.e. the states and transitions. In this section, the identification error for state merging is analyzed. Two types of errors can occur:

1. False negative (FN): an anomaly could not be found
2. False positive (FP): an anomaly is identified incorrectly

The Hoeffding Bound [Hoe63] will be used in several proofs. Therefore, it is introduced beforehand.

Lemma 3. *(Hoeffding's Inequality) Let $Z_1, ..., Z_n$ be independent, identically distributed random variables, with $0 \leq Z_i \leq 1$. Then*

$$P\left[\left|\frac{1}{n}\sum_{i=1}^{n} Z_i - E\left[Z\right]\right| > \epsilon\right] \leq \alpha = 2\exp\left(-2n\epsilon^2\right) \tag{7.2}$$

with ϵ the deviation between the empirical mean $\frac{1}{n}\sum_{i=1}^{n} Z_i$ and the true mean $E\left[Z\right]$, n the number of samples and a confidence $0 \leq \alpha \leq 1$ with $p = (1-\alpha)$ the probability of the decision.

That means, the probability that the deviation between the empirical and the true mean is greater than ϵ is less or equal than α.

7.2.1.1 False negative error

According to Hoeffding's inequality, we know that an observed transition probability $\frac{f}{g}$ is with a probability of $> (1-\alpha)$ at most $\sqrt{\frac{1}{2g}\log\frac{2}{\alpha}}$ away from the true probability p, where f is the number of observations for a specific event (incoming or outgoing transitions) and g is the sum of all observations for the corresponding state. Therefore,

$$P\left(\left|\frac{f}{g} - p\right| > h(g,\alpha)\right) = \alpha,$$
$$P\left(\left|\frac{f}{g} - p\right| \leq h(g,\alpha)\right) = 1-\alpha$$

with the Hoeffding bound $h(g,\alpha)) = \sqrt{\frac{1}{2g}\log\frac{2}{\alpha}}$.

Lemma 4. *For two observed transition probabilities $\frac{f_0}{g_0}, \frac{f_1}{g_1}$ and one corresponding probability p (i.e. both observed transition probabilities are from the same distribution), it follows that with a probability of $< 2\alpha$ (the confidence) the following holds*

$$\left|\frac{f_0}{g_0} - \frac{f_1}{g_1}\right| > h(g_0, g_1, \alpha)$$

with the bound

$$h(g_0, g_1, \alpha) = \left(\sqrt{\frac{1}{g_0}} + \sqrt{\frac{1}{g_1}}\right) \cdot \sqrt{\frac{1}{2}\log\frac{2}{\alpha}} \leq 2\alpha$$

This is the probability that a correct merging is not done by the algorithms.

Proof.

$$
\begin{aligned}
& P(|f_0/g_0 - f_1/g_1| > h(g_0, g_1, \alpha)) \\
=& P(|(f_0/g_0 - p) - (f_1/g_1 - p)| > h(g_0, g_1, \alpha)) \\
\leq& P(|(f_0/g_0 - p)| + |(f_1/g_1 - p)| > h(g_0, g_1, \alpha)) \\
\leq& P(|f_0/g_0 - p| > h(g_0, \alpha) \cup |f_1/g_1 - p| > h(g_1, \alpha)) \\
=& 1 - P(|f_0/g_0 - p| \leq h(g_0, a))P(|f_1/g_1 - p| \leq h(g_1, \alpha)) \\
=& 1 - (1 - \alpha)^2 \\
=& 2\alpha - \alpha^2.
\end{aligned}
$$

That means, at least one of the two error sources $\left|\frac{f_0}{g_0} - p\right|$ *or* $\left|\frac{f_1}{g_1} - p\right|$ *must have exceeded the error limit.* □

7.2.1.2 False positive error

False positive errors for the anomaly detection occur when a correct transition is missing in the learned automaton. This can only happen if a correct merging of two states has been missed by the algorithm:

Theorem 6. *The probability of a false positive error in the anomaly detection* p_{FP} *is bounded by* $1 - n^3|\Sigma| + 2\alpha n^3|\Sigma|$*, where* α *is the confidence,* Σ *is the set of symbols and* n *denotes the number of input samples.*

Proof. According to Lemma 4, the error for one call to the function compatible (see Algorithm 4) is bounded by 2α*. For each node pair,* $|\Sigma|n$ *such calls are needed since every node has a maximum of* $|\Sigma|$ *edges and the subtree is bounded by* n*. Furthermore, a maximum of* n^2 *state pairs can be checked for compatibility. This assumes independent tests. De la Higuera and Thollard have shown in [HT00] that dependencies only lower the probabilities, i.e. the errors are still bounded as shown above.*

So it follows that:

$$p_{FP} = 1 - (n^3|\Sigma|(1 - 2\alpha)) = 1 - n^3|\Sigma| + 2\alpha n^3|\Sigma|$$

□

False negative errors for the anomaly detection occur when an incorrect transition exists in the learned automaton. This can only happen if an incorrect merging of two states has been carried out by the algorithm:

So in the following line of argument, we assume that two observed transition probabilities $\frac{f_0}{g_0}$ and $\frac{f_1}{g_1}$ stem from two different distributions. We now have to compute the probability that is generated form different distributions, $\left|\frac{f_0}{g_0} - \frac{f_1}{g_1}\right| \leq \left(\sqrt{\frac{1}{g_0}} + \sqrt{\frac{1}{g_1}}\right) \cdot \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}$.

Lemma 5. *Let* $\frac{f_0}{g_0}$ *and* $\frac{f_1}{g_1}$ *stem from two different distributions and* $\frac{f_0}{g_0} \to p_0, \frac{f_1}{g_1} \to p_1$*. Then the probability of an incorrect merging is bounded by* 2α *iff* $p_1 - p_0 > 2\gamma, \gamma = \max(\gamma_0, \gamma_1), \gamma_i = \sqrt{\frac{1}{2g_i}\log\frac{2}{\alpha}}$*, where* α *is the confidence value.*

Proof. Incorrect merging happens if either one of the observations $\frac{f_0}{g_0}, \frac{f_1}{g_1}$ *lies within the intervals* $[p_0 - \gamma_0], [p_1 - \gamma_1]$ *of the other distributions. Figure 7.3 shows a situation on its left hand side whereat–correctly–a merging is not done.*

On its right hand side, the error scenario is shown. Here we first assume that $\frac{f_0}{g_0}$ *is incorrectly classified as stemming from* p_1 *and an incorrect merging happens.*

In the left-hand scenario, it must hold that $p_1 - p_0 > \gamma_0 + \gamma_1$*. And this inequality is true if* $p_1 - p_0 > 2\gamma$*.*

Under this constraint, an (incorrect) merging can only happen if one of the two observations $\frac{f_i}{g_i}$ *is outside its interval* $[p_i - \gamma_i]$*. And the probability of this is (see the explanations above)* 2α*.* □

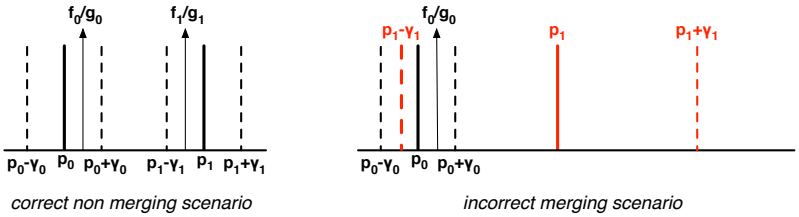


Fig. 7.3 Idea for proving Lemma 5.

Theorem 7. *If* $\forall i, j : p_i - p_j > 2\gamma$*, the probability of a false negative error in the anomaly detection* p_{FN} *is bounded by* $1 - n^3|\Sigma| + 2\alpha n^3|\Sigma|$*, where* α *is the confidence,* Σ *is the set of symbols and* n *denotes the number of input samples.*

Proof. According to Lemma 5, the error probability for one test error for false negative errors corresponds to the error for a false positive error (iff $\forall i, j : p_i - p_j > 2\gamma$*), i.e. we can repeat the idea of Theorem 6.* □

Remark 2. The constraint $\forall i, j : p_i - p_j > 2\gamma$ means that we must assume that probabilities occur only in specific steps. And the step size depends on the number of samples in the least used transition. In practice this is not a problem since the step size shrinks proportional to $\sqrt{1/n}$, if we presume that for each observation $\frac{f_i}{g_i}$ it holds that $g_i = cn, c \in \mathbf{R}$ where $n \in \mathbf{N}$ denotes the sample size.

Corollary 1. *If* $\forall i, j : p_i - p_j > 2\gamma$*, both the false positive error* p_{FP} *and the false negative error* p_{FN} *(see Theorems 6 and 7) can be kept below any given maximum error level* p_{max} *by choosing a specific value for* α*.*

Proof. If we set $\alpha = \frac{1-p_{max}}{2n^3|\Sigma|} - \frac{1}{2}$*,* p_{FP} *and* p_{FN} *will be bounded by* p_{max}*. To see this, insert this* α *into the error bounds of Theorems 6 and 7.* □

7.2.2 Timing learning error

In the same way as for state merging errors, the error for the timing identification is calculated. Three error sources are analyzed here in detail:

1. Wrong separation of a single mode into different modes
2. Wrong combination of different modes into one mode
3. Timing identification error in one transition

The computation of the combination and separation errors is based on the separation method, using the kernel density estimation as described in Section 6.2.3.3.

First, the definition of the correct timing behavior representation is given. Then, the analysis of the identification error of the algorithm follows.

7.2.2.1 Correct timing behavior

The timing information in one transition is given as probability density distribution (PDF) over time and a minimum and maximum time value. The PDF is calculated globally over all transitions that are using the same event (symbol a in the alphabet Σ), while the minimum and maximum time value is identified locally for each transition. Thus, it can happen that the mean value lies beyond the identified time range. However, this still belongs to the correct behavior. This case is illustrated in Figure 7.4.

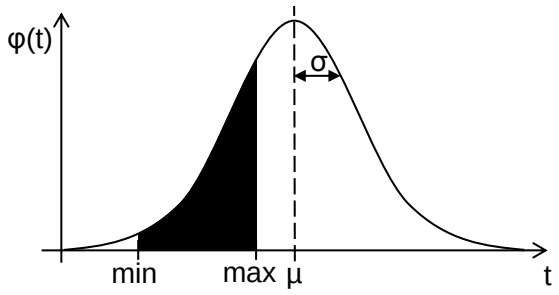


Fig. 7.4 Globally calculated PDF and locally identified time range can lead to the situation that the mean of the distribution lies beyond the identified time range.

During the timing preprocessing, it is globally checked for each event whether the time values are distributed in different modes. Events with multiple modes are separated into the single modes such that each transition consists of an event with the corresponding local time range and the global single-mode PDF. However, the separation of different modes is subject to errors. Errors occur either when different modes are wrongly combined into one mode or when a single mode is wrongly separated into different modes.

7.2.2.2 Wrong separation of a single mode into different modes

The error of wrong separation of a single mode into different modes mainly occurs when only few learning examples are available to identify the distribution functions. Figure 7.5 illustrates this error: The correct distribution with μ_{12} was not identified. Instead, two other distributions with μ_1 and μ_2 are identified.

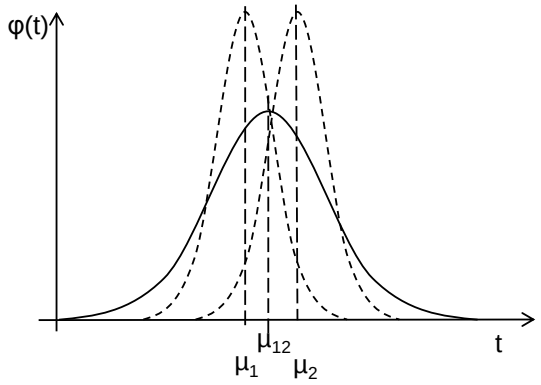


Fig. 7.5 One mode (with μ_{12}) is wrongly separated into different modes (with μ_1 and μ_2)

Theorem 8. *With a growing number of learning samples, the error of wrong separation of a single mode into different modes converges to* 0*.*

Proof. From the Hoeffding bound, we know that $\epsilon \leq \sqrt{\frac{1}{2n} \log \frac{2}{\alpha}}$*. From this it follows directly that with growing number of samples* n*, the error* ϵ *converges to* 0*.*

□

The error of wrong separation of a single mode into different modes leads to the situation, that two states can not be merged even if they are compatible. This error has been investigated in Section 7.2.1.

7.2.2.3 Wrong combination of different modes into one mode

The error of wrong combinations of different modes into one mode mainly occurs when two distributions are located closely to each other (small margin γ) and when the deviation from the mean is high (large standard deviation σ). An illustration of the error emergence is given in Figure 7.6: Two different distributions with μ_1 and μ_2 are wrongly combined into a PDF with one mode with μ_{12}.

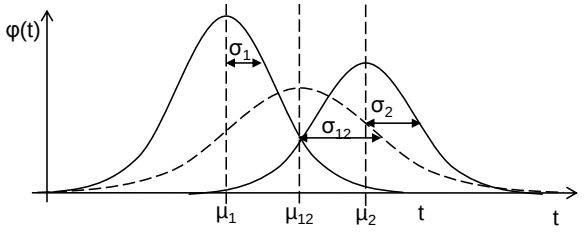


Fig. 7.6 Two different modes (with μ_1 and μ_2) are wrongly combined into one mode (with μ_{12})

Following factors are influencing the error: *(1)* margin between the means of the distributions ($\gamma = |\mu_1 - \mu_2|$), *(2)* spread of the values (σ_1, σ_2) and *(3)* the number of learning examples.

Assumption 2 *The calculation of the error assumes the following values to be known: (1) number of observations, (2) mean and spread of the distribution and (3) margin between the means of distributions.*

First, we assume that the PDF is been generated using an infinite number of learning examples, i.e. the PDF is calculated error-free. In a later step, the influence of the number of learning examples on the error is also investigated.

Furthermore, it is assumed that there exist two distributions that are overlapping. More than two distributions are handled in the same way.

The error is the uncertainty at the local minimum between two modes in the calculated PDF. According to Equation 6.1, the PDF for a data vector $\mathbf{t}$ is calculated as

$$f(\mathbf{t}, t, \mu, \sigma) = \frac{1}{N}\sum_{i=1}^{N} k(\mathbf{t}_i; t) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(t-\mu)^2}{2\sigma^2}}. \tag{7.3}$$

Analogous, the density of the sum of two different distributions is given as

$$f(\mathbf{t}_1, \mu_1, \sigma_1, \mathbf{t}_2, \mu_2, \sigma_2, t) = \frac{1}{N_1}\sum_{i=1}^{N_1} k(\mathbf{t}_{1,i}, \mu_1, \sigma_1, t) + \frac{1}{N_2}\sum_{i=1}^{N_2} k(\mathbf{t}_{2,i}, \mu_2, \sigma_2, t). \tag{7.4}$$

The local minimum between the two peaks is the separation point $t_{separation}$.

The error ϵ is calculated using the densities for the mean value and the separation point. It is expressed as the difference of the maximum (density of the mean value) and local minimum (density of the separation point) of the density function (see also Figure 7.7).

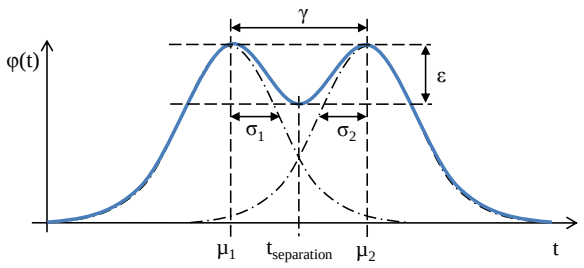


Fig. 7.7 Illustration of ϵ, the timing distribution separation error.

$$\epsilon = f(t = max(\mu_1, \mu_2)) - f(t = t_{separation}) \tag{7.5}$$

It can be distinguished between three cases, based on the margin γ:

1. $\gamma = 0$: $\epsilon = 1$, because it can not be distinguished between two completely overlapped distributions.

2. $\gamma = \infty : \epsilon = 0$, because there is no overlapping and the distributions can be separated completely.
3. $\gamma > 0$ and $f(t = t_{separation}) > 0$: $0 < \epsilon \leq 1$. From a theoretical point of view, a small margin between distributions is more interesting. Parts of the distribution functions are overlapping. This leads to an error. Figure 7.7 illustrates this case.

Lemma 6. *If $\gamma > \sigma_1 + \sigma_2$, the modes of two distributions with means μ_1 and μ_2 and standard deviations σ_1 and σ_2 can be separated correctly.*

Proof. For one PDF, it is known that the largest positive gradient is at the point $\mu - \sigma$, while the largest negative gradient is at the point $\mu + \sigma$.

Assuming two PDFs from independent random variables (generated with an infinite number of observations, i.e. error-free) with means μ_1, μ_2 and standard deviations σ_1, σ_2, where $\mu_2 = \mu_1 + \sigma_1 + \sigma_2$. Adding both PDFs, the maximum of the resulting PDF will be exactly at $x_{max} = \mu_1 + \sigma_1 = \mu_2 - \sigma_2$. [1] *This means that Algorithm 2 will not find a local minimum to create a separation point. The same applies for $\gamma < \sigma_1 + \sigma_2$.*

If $\gamma > \sigma_1 + \sigma_2$, a local minimum will be found, since the potential separation point $\mu_1 + \sigma_1 < x_{max} < \mu_2 + \sigma_2$.

□

The error further depends on the number of learning examples, since the correctness of the generated probability density function becomes more precise for a growing number of examples. This is captured using the Hoeffding bound. In Theorem 8 we have already seen that the error in creating the PDF shrinks with a growing number of learning samples.

The error of wrong separation of a single mode into different modes leads to the situation that two states can not be merged even if they are compatible. This error has been investigated in Section 7.2.1.

7.2.2.4 Timing identification error in one transition

Finally, an identification error can occur in single transitions. This happens when only few learning examples are available to identify the distribution function.

Assumption 3 *In the line of argumentation, it is assumed that the separation into the modes performed well, i.e. without any error. Furthermore, only uni-modal distribution functions are considered. Multi-modal distribution functions have already been separated in different events.*

Using Lemma 3, we can formulate the following theorem:

Theorem 9. *Let $Z_1, ..., Z_n$ be independent, identically distributed random variables, with $0 \leq Z_i \leq 1$. Drawing n samples, with probability at least $(1 - \alpha), \alpha \in \mathbb{R}$, the difference between the empirical mean $\frac{1}{n} \sum_{i=1}^{n} Z_i$ and the true mean $E[Z]$ is at most ϵ, where*

[1] This can be checked by creating the sum of both PDFs, calculating the derivation of the sum and determining the zero.

$$\epsilon \leq \sqrt{\frac{1}{2n} \log \frac{2}{\alpha}} \tag{7.6}$$

This ϵ directly corresponds to the bound of timing errors in a single transition.

Proof. Follows directly from Lemma 3.

□

Timing identification errors in one transition do not lead to a merging error.

7.3 Convergence and Identification In The Limit

Based on the error calculation from Section 7.2, the capabilities of *identification in the limit* are proven in this section.

If the number of examples is constant and if positive as well as negative examples are used, Gold proved in [Gol78] that the identification of a DFA of the given size $k \in \mathbf{N}$ is NP-complete, i.e. no efficient algorithms exist—under the assumption that $P \neq NP$. However, Oncina and Garcia showed in [OG92] that an efficient algorithm exists when the number of observations is not constant but can grow during an algorithm's runtime,—this is called identification in the limit. In this context, an algorithm is efficient when it has a runtime polynomial to the number of observations and the number of observations is polynomial to the size of the optimal automaton (recall Definition 19).

It has been proven that probabilistic DFAs [CO99, TDdlH00] and also probabilistic DTA with one clock [Ver10] are identifiable in the limit in polynomial time. Such probabilistic automata only use positive observations.

As long as a single clock is used (in fact only one clock is used), it can be said that BUTLA identifies the system behavior model in the limit in polynomial time. The polynomial runtime behavior has been proven in Lemma 1, so only the identification in the limit has to be shown. Two parts have to be proven:

1. States identification in the limit. This ensures that the automaton structure is identified correctly in the limit.
2. Timing identification in the limit. This ensures that the timing behavior (distribution functions and time ranges) is identified correctly in the limit.

7.3.1 States identification in the limit

Lemma 7. *If $\forall i, j : p_i - p_j > 2\gamma$, the algorithm BUTLA identifies a correct automaton in the limit.*

Proof. This directly follows from Corollary 1. □

The concept of "identification in the limit" has a crucial drawback: Nothing is said about the problem complexity in terms of data size n and in terms of runtime complexity. For this, two new concepts are introduced: 1. "Weak Polynomial Identification in the Limit with Probability One" requires that the algorithms has a runtime polynomial to the number of observations. 2. "Strong Polynomial Identification in

the Limit with Probability One" additionally requires that only a polynomial number of observations in the automation size is used.

Theorem 10. *If $\forall i, j : p_i - p_j > 2\gamma$, the algorithm BUTLA fulfills the criterion of "Weak Polynomial Identification in the Limit with Probability One".*

Proof. Follows from Lemma 7 and from Lemma 1. □

In [dlH10] de la Higuera argues that PDFAs are not strongly polynomially identifiable in the limit with probability one. Here, we show that if we give some limitations, the criteria of "strong polynomial identification in the limit with probability one" can be fulfilled.

Theorem 11. *If $\forall i, j : p_i - p_j > 2\gamma$ and if the automaton size $|\mathcal{A}|$ is larger than $|\Sigma|$, the algorithm BUTLA fulfills the criterion of "Strong Polynomial Identification in the Limit with Probability One".*

Proof. We already know from Theorem 10, that "Weak Polynomial Identification in the Limit with Probability One" is fulfilled.

The error probability of BUTLA is bounded by $c_1 \alpha n^3 |\Sigma|, c_1 \in \mathbf{R}$ (see Theorems 6 and 7). So, to achieve an error probability of $< \delta$, we set $\alpha = \frac{c_2}{n^4}, c_2 \in \mathbf{R}$. This fulfills point (a) from Definition 21.

From this, it follows that the criteria are fulfilled for $n \geq \frac{c_3|\Sigma|}{\delta}$. And since we assume that the automaton size $|\mathcal{A}| > |\Sigma|$ (this only means that there are no unused symbols in the alphabet), it follows that the criteria are also fulfilled for $n > \frac{c_3|\mathcal{A}|}{\delta}$. From this, it follows that points (b) and (c) from Definition 21 are fulfilled. □

7.3.2 Timing identification in the limit

This Section firstly introduces the concept of *Timing Identification In The Limit*. Similar to the known framework of *identification in the limit*, we calculate the timing identification error and prove that the error converges to zero for a growing number of learning examples.

Using the Hoeffding bound, the question can be answered that how many learning samples are needed to identify a correct automaton. This leads to the concept of *timing identification in the limit*.

Lemma 8. *Given a required accuracy ϵ and a confidence α, we need at least $n = \log\left(\frac{2}{\alpha}\right) \cdot \frac{1}{2\epsilon^2}$ samples.*

Proof. From the Hoeffding Bound we know that $\epsilon \leq \sqrt{\frac{1}{2n} \log \frac{2}{\alpha}}$. From this it follows directly that $n \geq \log\left(\frac{2}{\alpha}\right) \cdot \frac{1}{2\epsilon^2}$.

□

The following theorem summarizes the issues from above and introduces the concept of *Timing Identification in the Limit*.

Theorem 12. *Given an infinite number of observations n, the timing identification error converges to 0 and if for each two timing distributions for one event $\gamma > \sigma_1 + \sigma_2$, i.e. in the limit, a correct timed automaton is identified.*

Proof. From the Theorem 8 and Lemma 9 we know, that the timing error decreases with a growing number of learning examples. Furthermore, the "states identification in the limit" has been shown in Theorems 10 and 11. Together with Lemma 8, it follows that the timing can also be identified in the limit.

□

7.4 Top-Down vs. Bottom-Up

Previous algorithms for learning (timed) automata such as MDI [TDdlH00] Alergia [CO99] and RTI+ [Ver10] work in a top-down order. The order of comparison for potential merges starts with the root state of the prefix tree acceptor and its first descendent. The order of the states is defined by the lexicographic order of the shortest sequence of events leading to a final state (leaf) in the PTA. The algorithm BUTLA works in a bottom-up order (see also figure 7.8). Here, we start with the final states and proceed via the ascendents towards the root state. Because each compatibility check for states v, w comprises a recursive checking of both sub-trees of v and w, BUTLA has the advantage of handling smaller sub-trees near to the root of the original prefix tree acceptor.

The new bottom-up merging order used here means a significant speed-up of the algorithm compared to previous algorithms [TDdlH00, CO99, Ver10]. Figure 7.8 shows an example: On the left hand side, two states will be merged. Before the merging step, both sub-trees must be checked for compatibility, this takes $O(n + n')$ steps. After the merging step, the merged sub-tree comprises $n + n'$ states and has to be determinized, this again takes $O(n + n')$ steps. After this merging step, further merging steps are done within the sub-tree, and again, further recursive compatibility checks and recursive determinization steps are required.

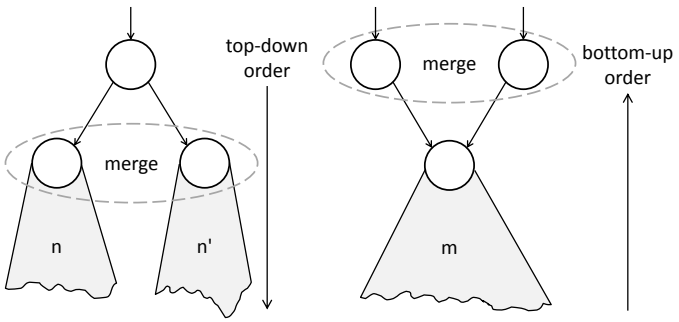


Fig. 7.8 The advantage of a bottom-up merging order.

The example on the right hand side shows the result of a bottom-up order merging: Since the sub-tree has already been merged in previous merging steps, the sub-trees share a large parts of their states and the compatibility can therefore be checked efficiently. Furthermore, no recursive determinization of the sub-tree is necessary because it was deterministic before. This is a rather optimist example, but it captures the gist of the bottom-up strategy.

In the worst case—no merging of states—this different order makes no difference for the runtime behavior. But in most cases, a significant runtime improvement can be seen whereat the degree of improvement depends on the structure of the prefix tree and on the structure of the final automaton.

Here we give a theoretical runtime comparison for an important class of problems: the complete tree problem. When observing cyber-physical production systems, we usually have sequences of, more or less, similar lengths—corresponding to a cycle of the cyber-physical production system, i.e. the prefix tree acceptor resembles a complete tree. And the final automaton usually resembles a sequence of states—corresponding to the states of the cycle of the run of the cyber-physical production system.

Definition 30. (Complete Tree Problem) In the Complete Tree Problem, the prefix tree acceptor is a complete tree of a fixed state out-degree of $k \in \mathbf{N}$. The optimal merged automaton is a single sequence of states.

Proposition 6. *For complete tree problems, top-down algorithms (using the state merging procedure as e.g. ALERGIA does) run in $O(n_0^2 \log n_0)$ whereas BUTLA runs in $O(n_0^2)$, where n_0 denotes the number of states in the original prefix tree.*

Proof. The $O(n_0^2 \log n_0)$ results from the sum of sub-tree sizes for all compatibility checks: $\sum_{i=1}^{n_0} \sum_{j=0}^{i} (\log(n_0 - i) + \log(n_0 - j))$ where the numbers correspond to the lexicographic state order. BUTLA merges two states in each compatibility check, so the subtrees always have the height 1. This results in a quadratic runtime behavior. □

The Complete Tree Problem is just an example for which the bottom-up technique performs better. On the contrary, there exist scenarios, where the top-down merging order performs better. This is especially the case for PTAs that contain just one long linked list of states.

In Section 8.3 we empirically analyze this scenario using a drunken sailor simulation, which creates a long linked list of states in the PTA.

Empirical measurements (see figure 7.9) show a rather $O(n_0^2)$ runtime behavior. Here, the Reber grammar was used again to compare the runtime. Both algorithms run in n_0^2 in average. However, the bottom-up algorithm works faster than the top-down approach by a factor of ≈ 3.

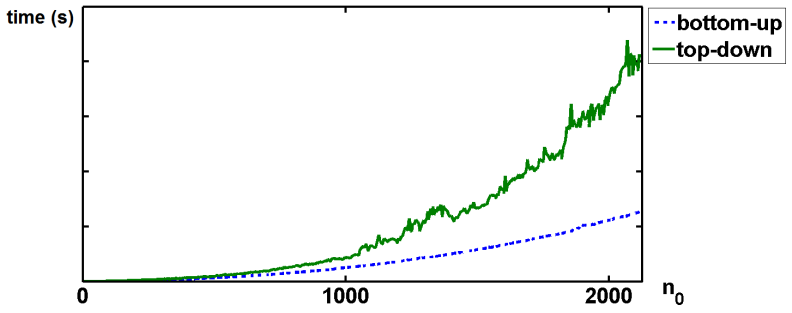


Fig. 7.9 Runtime behavior of the algorithms based on bottom up and top down strategy.

7.5 Splitting vs. Non-Splitting

Figure 2.8 illustrates that a state can be a starting point for different processes: When the robot is started, it depends on the size of the containers that which of the sub-trees is taken for the further process, based on the time that is needed to move the container. Different possibilities exist to identify the different timing behavior of the sub-trees.

The algorithm RTI+ proposed by Verwer in [Ver10] (see also Section 5.2.4) introduced a splitting operation for transitions. The algorithm calculates a p-value for all possible splits and its sub-trees. If the lowest p-value of one split is less than 0.05, the transition is split.

Figure 7.10 illustrates the problem of the splitting operation. The main drawback of using the splitting operation is that it requires additional computation time. First, all possible splits have to be evaluated. Based on the number of observations, these can be a huge amount. And after finding the best splitting point based on the smallest p-value, the transition has to be split. Here, for all postfixes of the corresponding transition, it has to be decided that which path to follow. Since all these paths are mixed in the previous states, the information that which path follows which states, based on the original data, has to be stored somehow. This leads to a huge memory consumption. To avoid this high memory consumption, RTI+ renews the prefix tree acceptor beginning with the corresponding state after each splitting operation. However, this is still time and space consuming.

Proposition 7. *The time complexity of calculating and performing a splitting operation is $\mathcal{O}(m^2 \cdot n^2)$, where m is the number of input samples and n is the number of states in the PTA.*

Proof. For each transition (in worst case there are $n-1$ transitions in the PTA, if it is a linked list of states with only one input sample or all input samples follow the path), the p-value has to be calculated (which has to be done for each input sample using the certain transition). Therefore, the complexity for calculating the p-values is $\mathcal{O}(m \cdot n)$.

One splitting operation itself also needs time in $\mathcal{O}(m \cdot n)$ for the creation of the PTA with m input samples, where each can have n states.

In the worst case, if each transition has to be split, the complexity is in $\mathcal{O}(m^2 \cdot n^2)$. □

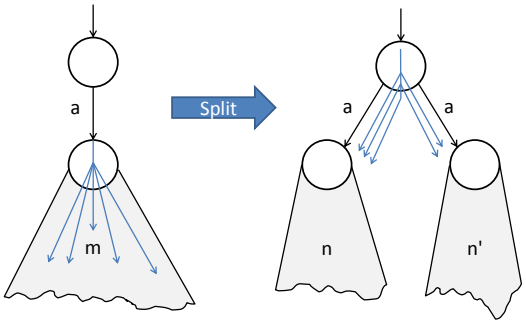


Fig. 7.10 The problem of the splitting operation.

BUTLA firstly uses a preprocessing of timing values to avoid this splitting operation. This version is based on the assumption that events with the same changing signals but different timing behavior describe different behavior.

In the preprocessing step, events with multiple timing modes are identified. These modes are used for the creation of the prefix tree. Events with the same symbol but arising from different timing modes are handled as different events and lead to different states in the prefix tree. In the identification phase, these events are also handled as different. Using this preprocessing step, the splitting process can be omitted. This leads to a computation speed increase.

Proposition 8. *The time complexity of calculating the timing modes in a preprocessing step is in $\mathcal{O}(n)$, where n is the number of observed events.*

Proof. Since this is during the preprocessing step and the PTA does not exist so far, the worst case is not dependent on the PTA structure, but only on the number of incoming events and the number of symbols.

First, the time stamps for each symbol in the alphabet $a \in \Sigma$ have to be collected. This takes time $\mathcal{O}(n)$.

Then for each $a \in \Sigma$, the probability density distribution over time has to be calculated. For this, Equation 6.4 is computed. Note that all events are not considered for a single symbol $a \in \Sigma$, but only those that belong to this symbol a. All computations together need time $\mathcal{O}(n)$. Additionally the local minimums have to identified, which is also done in $\mathcal{O}(n)$.

All these steps are performed subsequently and therefore the overall time complexity for the preprocessing step is $\mathcal{O}(n)$.

□

Using the preprocessing step, the computation time can be reduced compared to the splitting version. While the splitting version runs in polynomial time, we could reduce this additional timing computation to linear time using the preprocessing step.

7.6 Online vs. Offline Identification

This section compares the runtime behavior of both, online and offline automata identification algorithms, respectively OTALA and BUTLA. Both algorithms have different characteristics, so the runtime behavior can not be compared directly to assess which algorithm runs faster. Therefore, only a recommendation can be given that which identification algorithm should be used, based on the given use case.

As showed before, the identification time of OTALA is exponential to the number of input and output signals, whereas the runtime of BUTLA is cubic to the number of states in the prefix tree acceptor (the number of states in the prefix tree acceptor grows with the amount of observations).

BUTLA works best if the identification data is separated into cycles such that the prefix tree acceptor is highly condensed. However, this often cannot be guaranteed for cyber-physical production systems. In that case the prefix tree acceptor can consist of far more than one hundred thousand of states in a linked list. And since the identification time is cubic to the size of the prefix tree acceptor, this would lead to a high computation time.

In contrast, the runtime of OTALA is only based on the number of input and output signals and not on the amount of data. Therefore, the accumulated identification time over the input samples is rather linear. In cyber-physical production systems, the number of input and output signals is known beforehand. Therefore, the needed identification time can be roughly predicted. Mostly, the number of input and output signals is limited to a small number. This makes OTALA applicable, despite the exponential runtime.

Additionally, OTALA is able to stop the identification process if the identified model converged to the target model, such that all the data do not have to be considered for identification.

Summarized, comparing online (OTALA) and offline (BUTLA) identification algorithms, the following recommendations can be given:

The usage of an online identification algorithm (e.g. OTALA) is recommended, when:

- memory space is limited, such that the observations cannot be stored,
- it is not possible to use expert knowledge at all,
- the number of needed learning samples and the convergence of learning progress is not known beforehand or
- the observations can not be separated into cycles, which form a dense prefix tree acceptor.

On the contrary, the usage of an offline identification algorithm (e.g. BUTLA) is recommended, when:

- the number of signals is huge,
- the states of the target model can not be described by signal vectors and equal events can arise subsequently or
- the learning data can be separated into cycles.

Table 7.2 summarizes the comparison of the online identification algorithm OTALA with the offline identification algorithm BUTLA.

Table 7.2 Comparison of OTALA and BUTLA

	OTALA	BUTLA
not possible to use expert knowledge	+	-
limited memory space	+	-
amount of needed identification data is not known beforehand	+	-
observations can not be separated into cycles automatically	+	-
small number of signals	+	+
data can be separated into cycles	+	+
huge number of signals	-	+
states do not correspond to signal vectors	-	+

7.7 Runtime analysis of the anomaly detection algorithm

The anomaly detection is a time critical process. Similar to online identification, the observations cannot be cached but rather have to be evaluated directly. This has to be finished before the next event arises. Additionally, the decision whether an event belongs to the normal behavior or is an anomaly is required to be given promptly.

In Section 2.3.1, it was already mentioned that the anomaly detection in automata corresponds to the word problem in regular languages. The word problem is in $\mathcal{O}(n)$, where $n = |w|$ is the length of the word. We now show that this result can be refined:

Theorem 13. *The Anomaly Detection Problem (see Definition 9) using ANODA $P_{ANODA} \in \mathcal{L}$, i.e. the computation can be performed on logarithmic space.*

Proof. The automaton $\mathcal{A}$ is part of the input and is therefore stored on the input tape. The input string w is also stored on the input tape.

To show that a problem uses only logarithmic space, only the additional space has to be analyzed. Since the input and output tape need space at least linear to the input size, a logarithmic usage space were not possible.

At the beginning, the number of the initial state has to be stored (space: $log(n)$, where n is the number of states in the automaton). In each step, it is checked for each symbol (that is for each incoming event) whether there exists a transition in the automaton (stored on the input tape) with the actual event and the currently stored state as origin. Additionally the timing constraints are checked. This all happens in linear time and does not need additional space. If this is the case, the number of the corresponding destination state has to be stored. Since the last state number (which is currently stored on the working tape) is not needed any more, it can be overwritten. If no corresponding outgoing transition is found the input is rejected. At the end, there is the final state check. After reaching the last incoming event, it is checked whether the current state is a final state. If it is a final state, the input is accepted, otherwise it is rejected. This check does not need memory space and it only needs constant time.

The needed space is logarithmic to the size of the input (automaton $\mathcal{A}$ and input string w) because only one state number has to be stored and the binary encoding of this state number needs space logarithmic to the automaton size, i.e. the number of states in $\mathcal{A}$.

□

An empirical analysis of the needed time and real-time capabilities of the anomaly detection algorithm ANODA is given in [Vod13].

Chapter 8

Empirical Results

In this chapter, the Timed Automaton identification algorithms BUTLA and OTALA and the anomaly detection algorithm ANODA are evaluated empirically. In the last chapter, the algorithms have been evaluated theoretically. Here, we give some experimental results, which empirically confirm the results. The experiments are mainly based on artificial data and on real data, which was enlarged by simulation.

Since in [VMN13, Vod13], the algorithm BUTLA was compared with several other automata identification algorithms, we mainly focus on the comparison of BUTLA with OTALA.

The main contributions of this chapter are the following: we prove by empirical evaluation

- that BUTLA identifies a TA in polynomial time,
- that BUTLA identifies a correct TA in the limit,
- that OTALA converges to the correct TA,
- that OTALA on average runs much faster than the worst case runtime (which is exponential) such that it can be used for the identification of behavior models for cyber-physical production systems in acceptable time,
- that the bottom-up strategy works better than the top-down strategy in most recent cases and
- that ANODA (using identified Timed Automata) can be used for the anomaly detection in Cyber-Physical Production Systems.

This chapter is structured as follows: First, in Section 8.1 we explain which data sets we used for the empirical evaluation. Section 8.2 analyzes the runtime complexity of identifying Timed Automata. Section 8.3 evaluates the different influence of top-down and bottom-up merging order on the runtime behavior. In Section 8.4 the convergence behavior of BUTLA and OTALA is analyzed. Section 8.5 evaluates the capabilities of the anomaly detection algorithm ANODA. Section 8.6 evaluates the differences of online and offline identification algorithms (BUTLA and OTALA). Finally in Section 8.7, BUTLA is tested against two alternative approaches.

8.1 Preliminaries

This section leads to the empirical analysis of the automaton identification algorithms. Section 8.1.1 introduces some artificial data and a method to artificially extend real data . In Section 8.1.2 some quality measures are introduced, which are used to judge the experimental results.

8.1.1 Used Artificial Data

All researchers in data mining face the same problem: The lack of convincing amount of data from real systems. For this reason, artificial data is needed. This Section describes three possibilities of how to create enough data artificially: *(1)* The usage of artificial data (Section 8.1.1.1 and 8.1.1.2) and *(2)* the usage of a (learned) model to create an extended data set (Section 8.1.1.3).

8.1.1.1 Reber Grammar

One frequently used data set, which serves as benchmark data to compare the quality of grammatical inference algorithms, is the so called Reber grammar [Reb67]. Figure 8.1 shows the original automaton of the Reber grammar. Each transition probabilities are normally distributed. The probabilities are not displayed in Figure 8.1.

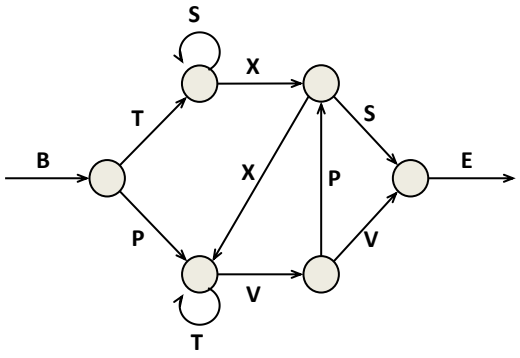


Fig. 8.1 Automaton of original reber grammar

Using the automaton for the Reber grammar, accepted words can be created easily. "BPVPXVPXVPXVVE" is an accepted word for example, and "BTSSPXSE" is not accepted. An accepted word can simply be created by simulating the automaton. To obtain a data set with n samples, the automaton just has to be simulated n times. Based on this data set, an automaton can be identified. Ideally, the original and the identified automaton should be equal. A list of non-accepted words can be used to check the quality of the identified automaton. Each word which is not accepted by the original should also not be accepted by the identified automaton.

Originally, the Reber grammar was created for benchmarks with stochastic automata. The transitions only contain symbols that are added one after the other and result in an accepted word. Events in automation systems describes the switching of an actuator or sensor, e.g. the starting of a conveyor or the switching of position sensor. Therefore, the symbols in automata consist of the name of a sensor/ actuator and the new signal value, e.g. "conveyer=1". Furthermore, the transitions in the Reber grammar have no timing constraints. To enhance the testing possibilities, arbitrary timing information was included for each transition. The timing information is given as a time range with the minimum and maximum relative time stamp.

Since the presented algorithms (BUTLA and OTALA) are used in automation systems, loops cannot be performed as they are present in the Reber grammar. Each event in a transition describes an event in an automation system. A Sensor or actuator cannot be switched on twice without being switched off in the meantime. The loops in the Reber automaton are modified in a way that each signal is switched off before being switched on again. This leads to three additional states.

The transition probabilities are not modified. In Figure 8.2, they are not displayed again.

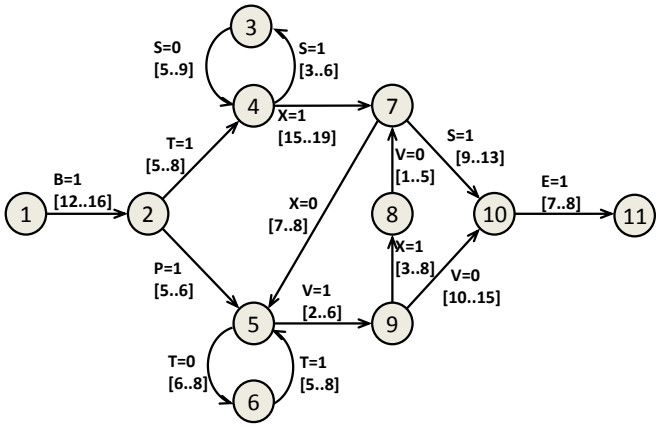


Fig. 8.2 Automaton of modified reber grammar

For the following experiments with the Reber grammar, only the modified version is used.

8.1.1.2 Random Walk

The random walk (also known as drunken sailor simulation, see also Figure 8.3) [Pea05] is a simulation method, which is based on pure randomness.

The method is known from graph theory. Here it is used as follows. At the beginning there is an initial state vector (all zeros or random combination). In each simulation step one random signal is changing its value. By running the simulation over some time, each state in the discrete state space will be visited.

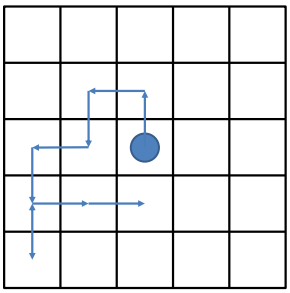


Fig. 8.3 Principle of the random walk (drunken sailor) simulation.

Since the random walk explores the whole (discrete) state space, it is well suited for worst case simulations. In this chapter, the random walk simulation is used for convergence and worst case runtime tests with OTALA.

8.1.1.3 Data from simulated model

This method gives the possibility to obtain a high amount of data, which is referred to a real system. Based on an existing model, many simulations can be carried out and the simulation outputs are stored in a database. These data are used afterwards to learn a behavior model.

Figure 8.4 shows how data can be manifolded and how the extended data can be used:

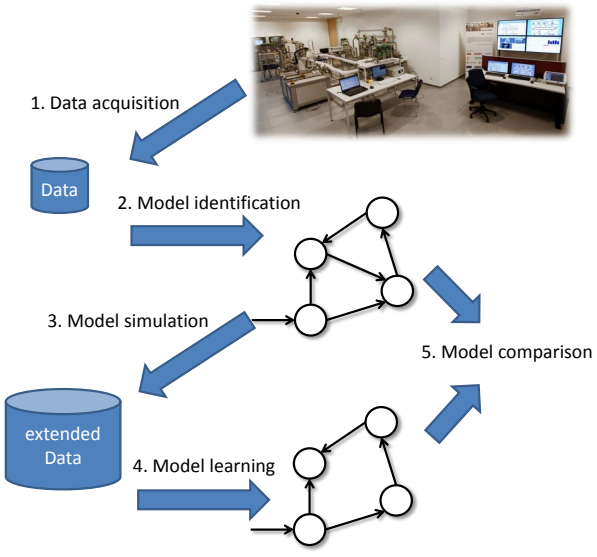


Fig. 8.4 Simulation of identified model to obtain more data.

1. **Data acquisition:** First, the production plant is observed and the data is stored into a data base.
2. **Model learning:** The acquired data is used to learn a model.
3. **Model simulation:** Usually, only few observed data exists. Therefore, the learned model from step (2) is simulated. Each needed information is available. Beginning with the starting state and based on the transition probabilities, it is decided that which state is visited next. The timing in the corresponding transition is chosen randomly according to the distribution function. Each simulation step, i.e. each generated event and the corresponding time information is logged into a data base. The simulation is performed until it stops in a final state. The simulation can be repeated as many times as needed. Many repetitions result in a significantly extended data set.
4. **Model learning:** Using the extended data set, a new model is learned.
5. **Model comparison:** Now, for evaluation purposes, both models can be compared.

In general, it is difficult to evaluate the correctness of an identified discrete model. Only experts of a plant know all the details of the normal behavior of a plant. Even for the experts, it is at least time consuming. For laymen, it is impossible, since they have no knowledge about the original behavior of the plant. On the contrary, using the aforementioned approach, the original model is available, which is the identified model from step (2). The original automaton and the identified one can now be easily compared.

8.1.2 Correctness of identified Timed Automata

The correctness of a classifier or model in general is evaluated using certain quality measures. Commonly used quality measures for instance are accuracy, precision or f-measure. Most of these quality measures are based on the computation of different ratios between correctly and incorrectly classified samples according to the confusion matrix. We used the confusion matrix according to [TSK06], which is shown in Table 8.4. It gives an overview over the number of True Positive, False Positive, False Negative and True Negative samples. These numbers are explained as follows:

- True Positives (TP) are samples that are really anomalous and detected as such by the anomaly detection algorithm.
- False Positives (FP) are samples that are free of anomalies, but recognized as anomalous.
- False Negatives (FN) samples are anomalies that are not detected by the algorithm.
- True Negatives (TN) are really normal samples that are recognized as such by the algorithm.

		Predicted Class	
		Positive	*Negative*
Actual Class	*Positive*	*True Positive* (TP)	*True Negative* (TN)
	Negative	*False Positive* (FP)	*False Negative* (FN)

Table 8.1 Confusion Matrix

The *Specificity* (or true negative rate) gives the probability that the algorithm signals no error, if the test sample is indeed not anomalous.

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

The *Sensitivity* (or true positive rate, recall) gives the probability that the algorithm detects an error, if the test sample is really anomalous.

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

The *Accuracy* is a measurement of the anomaly detection capabilities and is calculated as the ratio of the number of correct predictions and the total number of predictions:

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

Often it is difficult, or even impossible, to generate negative test samples. In the domain of cyber-physical production systems, this would require some records of production runs with inserted faults. However, this is not feasible since it is not possible to enforce all possible errors because not all possible faults can be foreseen. If only positive examples are available, the k-fold cross validation [RN10] can be used to divide the learning examples in a learning set and a test set. The overall training set is divided into k groups, of which $k-1$ groups are used for identification and the remaining group is used for testing.

Two special cases of the k-fold cross validation are very popular and also used later in this thesis: For the *2-fold cross validation*, the learning examples are divided into two groups. First, one group is used for model identification and the other group for testing, then the other way around. Using the *leave one out cross validation* (LOOCV), each of the k groups consists of only one sample, i.e. $k = n$, where n is the number of input samples.

8.2 Runtime Analysis of the Identification Algorithms

In Section 7.1.1, we analyzed the runtime of both, BUTLA and OTALA formally. In this section, we give some empirical results to show the runtime behavior using artificial and real data.

8.2.1 Runtime Analysis of BUTLA

Lemma 1 showed that the algorithm BUTLA runs in $O(n_0^3)$, where n_0 denotes the number of states in the PTA. However, the average runtime is rather quadratic. Figure 8.5 shows the result for the Reber grammar. Since the PTA of the Reber grammar contains only few states, the creation of the PTA was modified in such a way that each data example leads to a separate branch. This leads to a PTA with a huge number of states, depending on the amount of input samples.

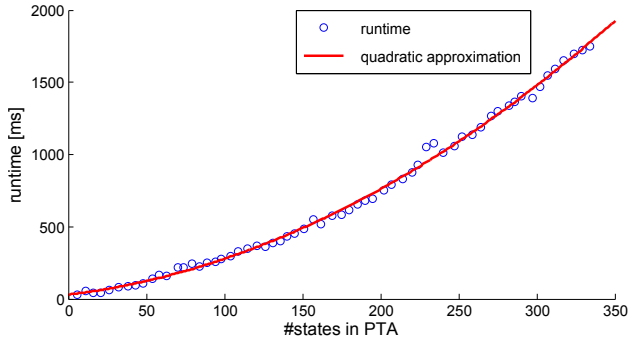


Fig. 8.5 BUTLA shows a rather quadratic runtime behavior for Reber data.

By computing the runtime in the same way as Carrasco did (based on the input sample size [CO94]), we obtain a rather linear behavior. However, these results can only be generated empirically and have no theoretical impact. The reason for the linear runtime behavior has been investigated in Section 7.1.1. Figure 8.6 shows the linear runtime behavior for an increasing amount of input data from the Reber grammar.

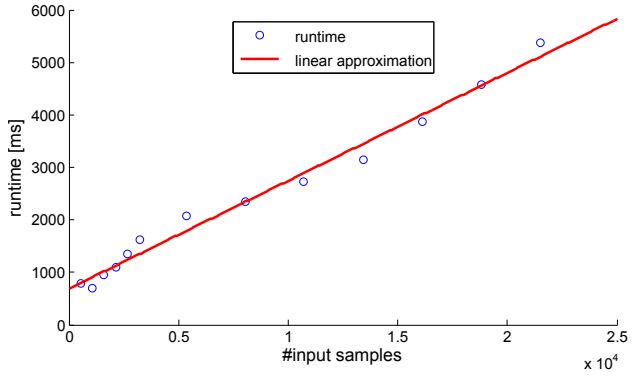


Fig. 8.6 Based on the amount of input samples, BUTLA shows a rather linear behavior.

8.2.2 Runtime Analysis of OTALA

In Lemma 2, it was shown that the runtime of OTALA is exponential in the number of IOs.

In general, this means that it is not suited for systems with a high amount of IOs. Nevertheless, according to [MNV+11] and [RSLL12] it is not useful to identify one overall model for the whole system but rather for subsets of the IOs. It means that only those signals are used to identify a model that belong to the same sub-process. This reduces the number of used signals again and makes OTALA applicable. Since the

runtime is dependent on the number of IOs, the worst case runtime can be determined beforehand.

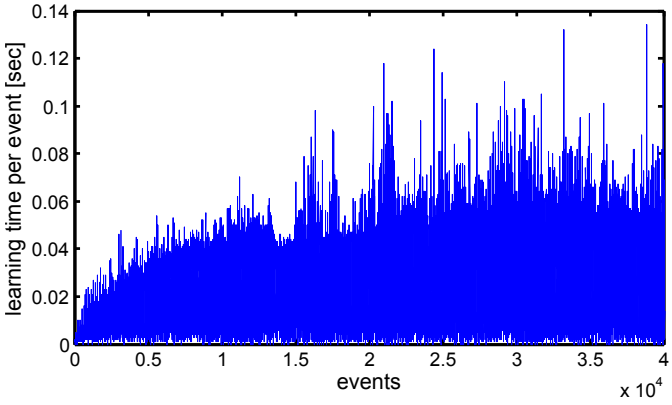


Fig. 8.7 Runtime behavior of OTALA.

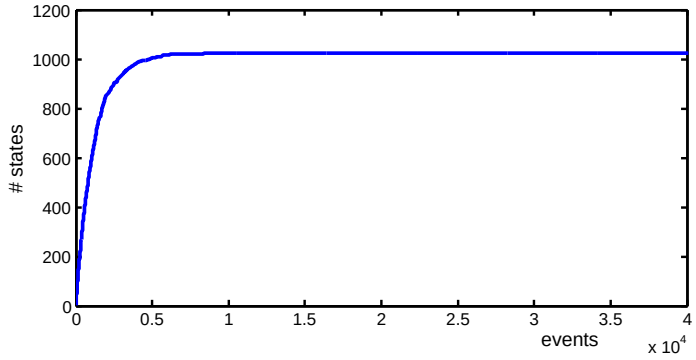


Fig. 8.8 Convergence behavior of OTALA.

We used the drunken sailor simulation with 10 signals for the worst case analysis. Figure 8.7 shows the computation time of OTALA for each incoming event. At the beginning of the incoming event sequence (approximately the first 1000 events), the computation time is rising and then remains approximately at the same level. This is due to the number of states, which is $2^{10} = 1024$. Comparing Figure 8.7 and Figure 8.8, it can be seen that the needed time rises as long as the number of current states increases. As far as the maximum number of states is reached, the runtime also remains approximately on the same level.

To examine this dependency in more detail, Figure 8.9 shows the runtime of OTALA plotted against number of states. The results are based on a data set created by a drunken sailor simulation with 20 signals. The target model comprises $2^{20} \approx 1$ million states. However, the simulation was aborted after 100000 events, such that the experimental results are given until about 90000 states. On the x-coordinate, the

number of states is given and on the y-coordinate, the individual needed time for finding the corresponding state.

It can be seen that the runtime can be approximated linearly to the number of states.

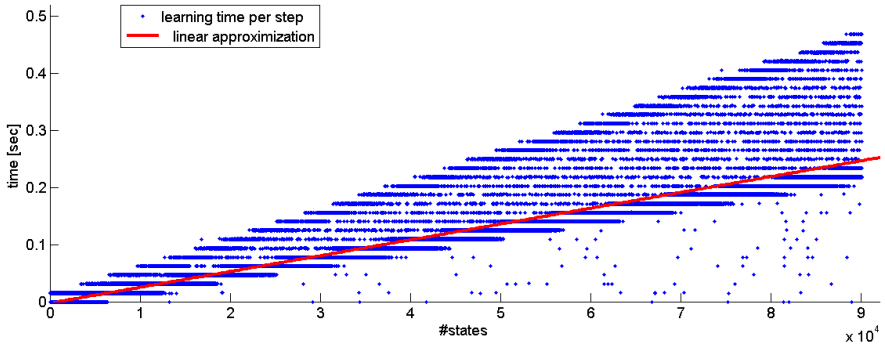


Fig. 8.9 Runtime of OTALA plotted against number of states.

Due to the linear dependency of the runtime on the number of states, it can be concluded that OTALA can be efficiently used to identify timed automata of cyber-physical production systems if it can be roughly estimated that the potential number of states is rather small.

Note that this experiment is based on an artificial data set (random walk) and a huge number of states is considered. According to Figure 8.9, approximately half a second is needed to find the correct state for a model with 90000 states and to create a new one if no appropriate state was found. However, the number of states in real plants rarely exceeds a few hundred. Additionally, the experiments are performed using MATLAB on a commercial practice PC. Using embedded programming on embedded devices, the computation speed can be further increased.

The runtime can be further reduced, if more memory space is used. For each possible state, memory space is allocated and the state ID corresponds to the decimal representation of the IO-vector. During runtime of the algorithm, it is not necessary to iterate over all possible states, just the IO-vector has to be converted into a decimal number and the corresponding memory space has to be addressed. The drawback of this method is the high amount of needed memory space and in usual cases, it is not used at all. Since the number of IOs is known beforehand, the needed amount of memory space can be predetermined.

The difference between the space and time optimized version of OTALA can be seen in figure 8.10. Here, the cumulative runtime over all incoming events is used. Both show a rather linear runtime behavior, the time optimized version runs about 10 times faster. The time optimized allocates the maximum amount of space at the beginning, even if it is not needed, whereas the space optimized version only allocates memory space when it is necessary.

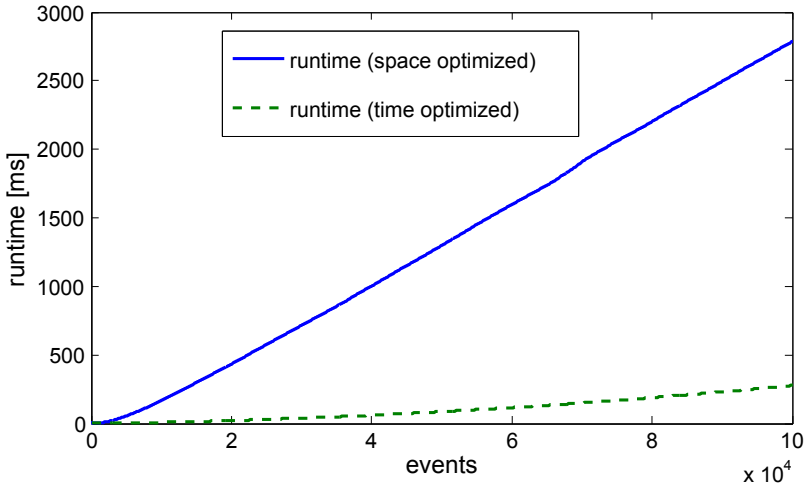


Fig. 8.10 Difference between the space and time optimized version of OTALA.

8.3 Top-Down vs. Bottom-Up

In this section, we empirically analyze the influence of the merging order (bottom-up and top-down) on the performance of the identification. For this, the same implementation is used, but the merging order is exchanged. Both, the bottom-up and top-down strategy are used to compare the influence on the learning procedure, without the effects that depend on other algorithms.

The analysis comprises the comparison of:

1. number of needed compatibility checks
2. number of (recursive) determinizations
3. runtime of the learning procedure

8.3.1 Experiments with the Reber grammar

First, we used the Reber grammar for some experiments. The modified Reber grammar, as shown in Figure 8.2, was used to generate input learning samples. These generated input samples were used to identify Timed Automata using both, the top-down and bottom-up strategy. Then we compared the three criteria from above. The results are given in Figures 8.11, 8.12 and 8.13.

Figure 8.11 shows the number of needed compatibility checks as a function of the number of states in the prefix tree acceptor n_0. It can easily be seen that the bottom-up strategy needs less compatibility checks. Due to the bottom-up merging order, less (in this case even no) recursive checks have to be performed than using the top-down strategy and therefore the amount of compatibility checks is lower.

Figure 8.12 shows the number of determinizations depending on the number of states in the PTA. While the number of determinizations for the top-down strategy is growing with the number of states in the PTA, the bottom-up strategy did not use any determinization. This is due to the small size of the input strings (which are around

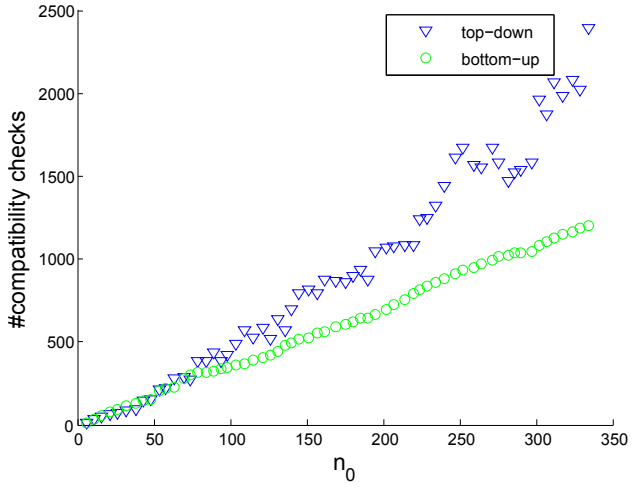


Fig. 8.11 Comparison of needed compatibility checks for BUTLA and OTALA using Reber grammar.

5-10 characters), which results in a small tree size. Since the bottom-up strategy starts with the leafs (final states) and proceeds upwards to the root (initial state), and the tree height is small, no recursive checks and, therefore, no determinizations are necessary.

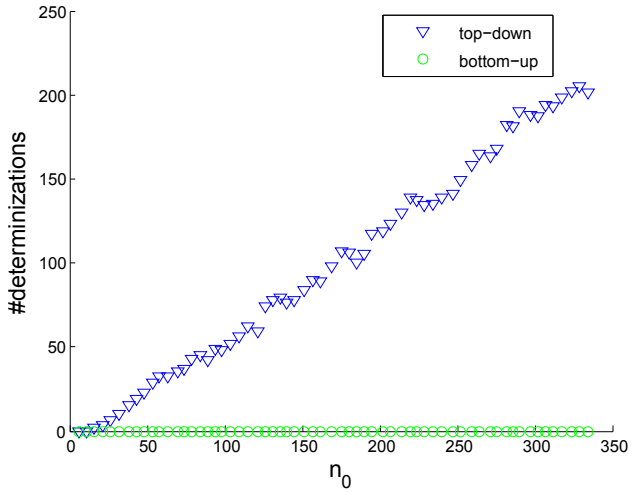


Fig. 8.12 Comparison of needed determinizations for BUTLA and OTALA using Reber grammar.

The number of needed compatibility checks and determinizations finally influences the needed amount of time. Since the bottom-up strategy needs less compatibility checks and less determinizations, the computation is also faster. Both, runtime for bottom-up and top-down strategy can be approximated quadratically, but the bottom-

up strategy works faster by a factor of ≈ 2.5. The runtime results are illustrated in Figure 8.13.

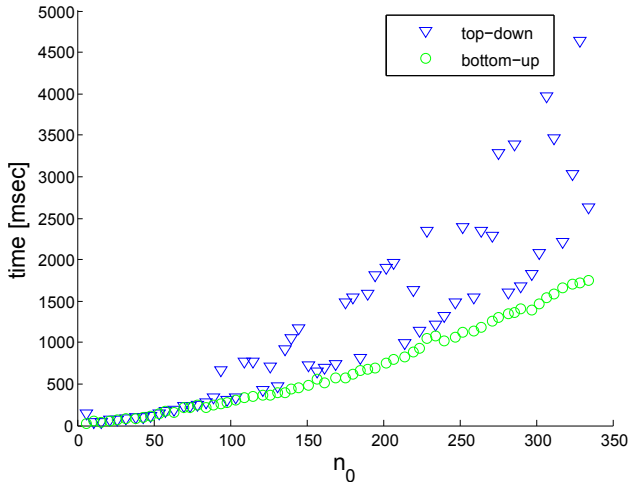


Fig. 8.13 Comparison of needed amount of time for BUTLA and OTALA using Reber grammar.

8.3.2 Experiments with an artificial data set based on random walk simulation

We also used the random walk simulation according to Section 8.1.1.2 to generate artificial data. The random walk simulation starts with a random initial condition (here: random signal vector) and runs randomly trough the whole state space. The simulation result is one long sequence of events. This leads to a PTA that consists of one long linked list of states.

In Section 7.4, we stated that the bottom-up strategy weakens for PTAs, which consist of only one long linked list of states. The advantage of the bottom-up strategy can only be exploited when there exist pairs of long compatible subtrees near to the leaf. However, the following two points are not met by the random walk simulation: *(1)* The random walk simulation provides only one long event sequence, i.e. no pairs of long subtrees and *(2)* the event sequences are generated by pure randomness and therefore only few compatible event sequences will be found such that the advantage of avoiding the recursive check cannot be exploited.

As can be seen in Figures 8.14, 8.15 and 8.16, the top-down merging order gives better results for this kind of PTAs.

Figure 8.14 shows the comparison of the number of compatibility checks. The bottom-up merging order needed significantly more compatibility checks than the top-down strategy.

As already mentioned for the Reber grammar, the top-down order performed significantly more determinizations. Approximately 1100 states (of the 1500 states in the PTA) have been merged in the determinization procedure.

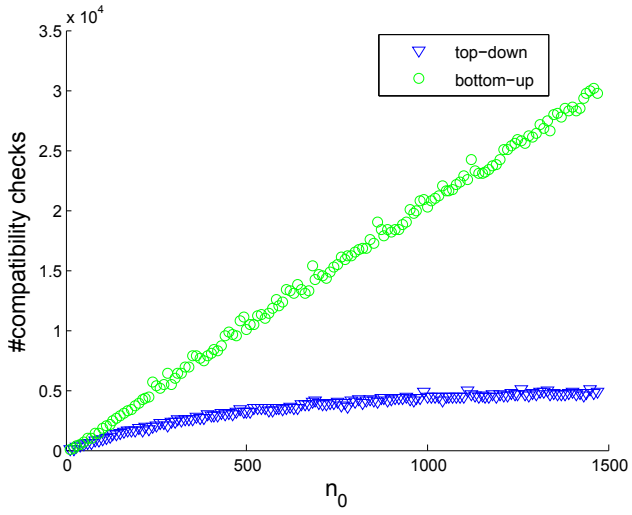


Fig. 8.14 Comparison of needed compatibility checks using data from the random walk simulation which leads to a long linked list as PTA.

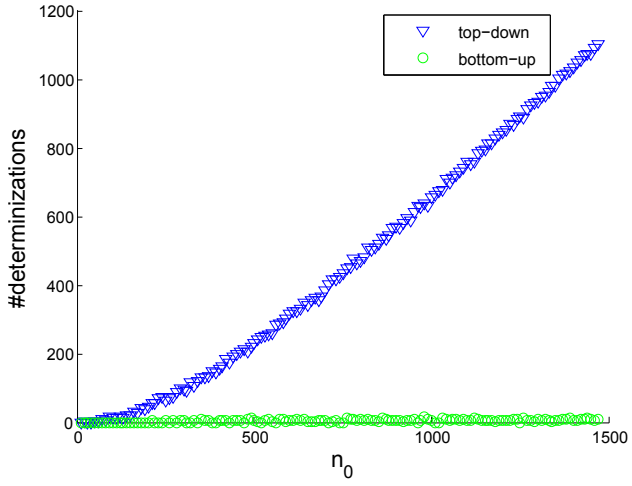


Fig. 8.15 Comparison of needed determinizations using data from the random walk simulation which leads to a long linked list as PTA.

Due to the high amount of determinizations in using the top-down merging order and the high amount of compatible checks using the bottom-up merging order, the bottom-up strategy requires more computation time for this kind of data. Figure 8.16 shows the simulation results for the computation time.

The results of the random walk simulation show the weakness of the bottom-up strategy. However, it is an artificial data set, which is far away from practice. The relevant application scenarios show a more structured behavior. In general, there exist pairs of long compatible subtrees, such that the advantages of the bottom-up strategy can be exploited.

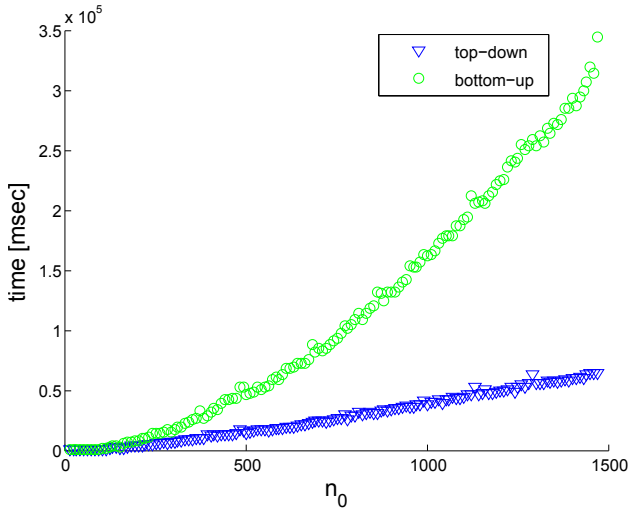


Fig. 8.16 Comparison of needed time using data from the random walk simulation which leads to a long linked list as PTA.

Finally, it can be concluded that the bottom-up strategy works faster for the relevant cases.

8.4 Convergence of the Identification Algorithms

Both, BUTLA and OTALA, converge towards the target automaton after a certain amount of input data. However, both algorithms show a different convergence behavior. OTALA is able to detect the convergence autonomously, while BUTLA is not able to do so. The convergence of BUTLA can only be shown by identifying several models and comparing them.

In this section, we analyze the convergence capabilities of the identification algorithms BUTLA and OTALA empirically.

8.4.1 Convergence of the Identification Algorithm BUTLA

In contrast to OTALA, BUTLA is not able to determine the convergence of the identification process. The convergence can only be determined a posteriori. For this, several input data sets with an increasing amount of data are used to identify multiple models and to compare the output model. If the models are different, a new data set with more input samples is used to identify a new model. At some point in time, the model will not change compared to the previous one. If it does not change further over a certain time, the identification process can be considered as converged.

In [Vod13], the number of identified states is taken as convergence criterion. Here, we extend the convergence criterion using a qualitative measure.

The correctness of a model can be determined by its ability to recognize anomal behavior. Therefore, in an iterative manner, models with an increasing number of input samples are identified. The identified models are then used for anomaly detection using a separate data set with positive samples that are not used for model identification. At some point in time, the specificity (true negative rate) should reach 100%, i.e. all positive samples are recognized as such by the algorithm, which means that the identification converged. If a representative data set is used, the identification process can be considered as converged as soon as the specificity reaches 100%.

In an experiment, the Reber grammar was used to evaluate the convergence behavior. As identification input, samples from the Reber grammar was used in increasing number. The anomaly detection was performed on a separate data set of 200 positive samples, which were not used for model identification. Figure 8.17 shows the convergence curve of the specificity. After training the model with 130 or more learning samples, the specificity reached 99.5%, only one of the 200 samples could not be classified correctly.

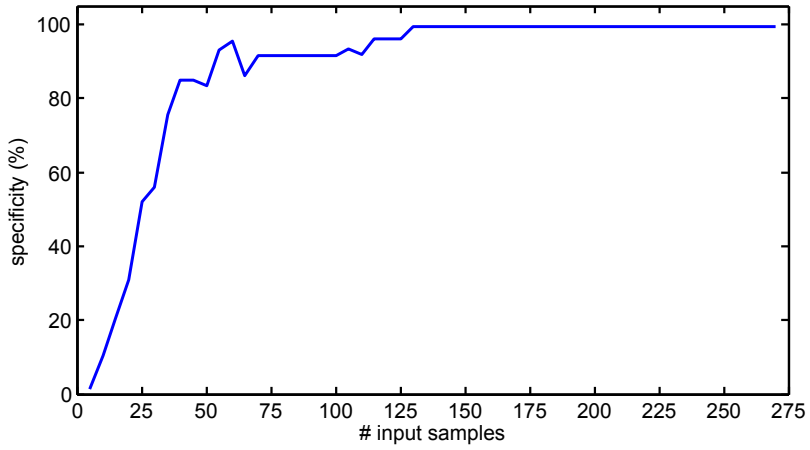


Fig. 8.17 Convergence of the model identification with BUTLA based on specificity calculation.

Of course, this method of analyzing the convergence of identification using BUTLA is only usable in offline experiments and not for the identification in practice. But it shows the ability of BUTLA to converge to the correct automaton.

8.4.2 Convergence of the Identification Algorithm OTALA

In Section 6.3, we introduced the online identification algorithm OTALA. One of its main benefits is that it is able to recognize the convergence of the identification by checking whether the input samples lead to a change in the model. If the model did not change for a certain amount of time, the identification process can be considered as converged.

In this section, the convergence of OTALA is empirically analyzed. According to Section 6.3, we use the following convergence criteria:

1. Number of states
2. Number of transitions
3. Number of changing (enlarging) time bounds

In Section 6.3, it was claimed that the correct model is identified when the aforementioned criteria converged. In this section, we empirically analyze this convergence of the model identification using the method as described in Section 8.4.1: Step by step, in an iterative manner, a model is identified for a given data set using an increasing amount of learning samples. In each step, a test data set of 200 positive examples, which were not used for identification, was used for anomaly detection. For each setting, the specificity according to Equation 8.1 is calculated. This is compared with the convergence of the three mentioned criteria.

The two curves are displayed in Figure 8.18

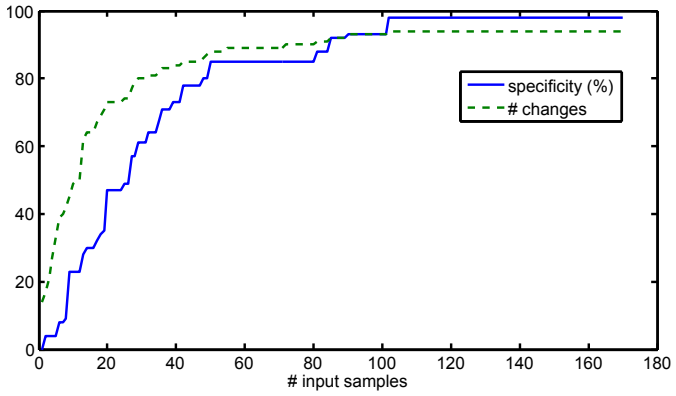


Fig. 8.18 Comparison of the convergence curve (number of changes) with specificity for OTALA.

Both, the curve of calculated specificity values and the curve of amount of changes, show a similar behavior. At around the 100^{th} input sample, both reach their final value. This means that the three aforementioned points can be used to determine the convergence of identification.

8.4.3 Comparison of the Convergence Behavior with BUTLA and OTALA

In this section we compare the convergence behavior of the identification process with BUTLA and OTALA. Using the same data set (in an iterative manner with an arising amount of learning examples), both algorithms were used to identify Timed Automata. For each identified automaton, a disjunctive data set of 200 positive learning examples was used to perform the anomaly detection, again the same data set for BUTLA and OTALA. As a convergence criterion, the calculated specificity is used as before. To compare the convergence behavior, both convergence curves are plotted in the same figure (see Figure 8.19).

It can be seen that both algorithms show a similar convergence behavior. OTALA needed approximately 100 learning samples to reach the final specificity of 98,5%.

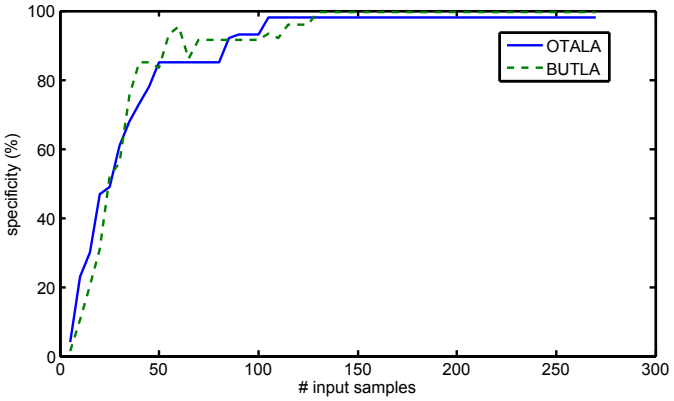


Fig. 8.19 Comparison of the convergence behavior using BUTLA and OTALA.

BUTLA needed some more learning examples (around 130) to reach the final specificity of 99.5%, which however was slightly better than OTALA.

Since both algorithms show a similar convergence behavior, the automatic convergence detection of OTALA can be exploited when determining the needed amount of data for the identification with BUTLA. For this, the identification algorithm OTALA runs in parallel during data acquisition (see step 0 in Section 6.2.1). As soon as OTALA converged (the three criteria from Section 6.3), it can be assumed that the amount of data is sufficient to converge with BUTLA as well.

8.5 Empirical Analysis of the Anomaly Detection

As already mentioned in Section 6.2.3, the timing is analyzed in a preprocessing step for each event. In this context, the timing distributions in each event are identified. If the timing behavior consists of several different distributions, these are identified as well.

The separation into different modes is not always performed correctly. Following errors can occur in the identification of the underlying distributions:

1. Correctly identified single-mode distribution but errors beyond the borders of the minimum and maximum time value
2. Two modes are separated correctly but parts of the distributions are overlapping

These error types are evaluated in this section. Since the identification of the underlying distribution functions assumes a Gaussian distribution (see Section 6.2.3.2), here, the calculation of the identification error is based on the same assumption. The same way it can be calculated for the assumption of other distribution functions.

In Section 8.5.3 we empirically analyze how the anomaly detection accuracy can be increased by enlarging the identified time bounds.

8.5.1 Error beyond the borders of the minimum and maximum time value

False positive timing errors occur when a timing is observed for a correct behavior during runtime that has not been learned before and therefore is not included in the model. ANODA uses the learned relative time ranges to check the timing behavior, i.e. a timing error is signaled if the observed time value is below the minimum or above the maximum time stamp.

To calculate the false positive timing error for a single transition, again, a normal distribution for the relative timestamps is assumed. Given the density function

$$\phi(t) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}, \tag{8.4}$$

with μ the mean and σ the standard deviation of the underlying normal distribution for the observed data, the false positive error P_{FP} can be calculated as

$$P_{FP} = \int_{0}^{min} \phi(t)dt + \int_{max}^{\infty} \phi(t)dt. \tag{8.5}$$

The error P_{FP} is marked in figure 8.20.

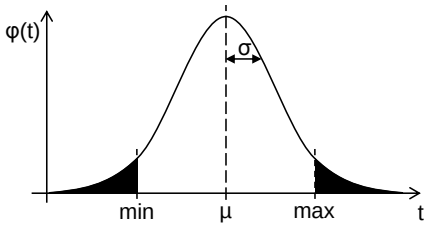


Fig. 8.20 Gaussian normal distribution and the false positive error (black area)

Empirical measurements delivered an average false positive rate of about 3%, however, if there exist only few observations for a certain branch in the automaton, the false positive error for single transitions can rise up to 30%.

The false positive error rate could be reduced using wider time ranges, but the false negative error rate raised in the same time.

Remark 3. The normal distribution is not restricted to positive values. However, time stamps in technical systems can only have positive values. Yet this calculation still holds true since the error, as given in Equation 8.5, only gives an upper bound. The real error is always smaller.

8.5.2 Error with overlapping distributions

So far, only the case of uni-modal distribution has been considered. Taking multi-modal distributions into account, the overlapping error has to be considered as well.

Assuming two Gaussian distribution functions according to Equation 8.4, the overlapping error can be calculated as

$$P_{FP} = \int_{0}^{min_2} \phi_2(t)dt + \int_{max_1}^{\infty} \phi_1(t)dt, \tag{8.6}$$

where $\phi_1(t)$ and $\phi_2(t)$ are the two identified distribution functions with their parameters respectively. Additional to the overlapping error in Equation 8.6, the error beyond the borders of the minimum and maximum time value has to be calculated according to Equation 8.5.

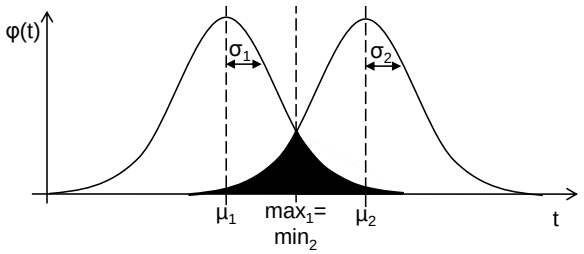


Fig. 8.21 Gaussian normal distribution and the overlapping of two distribution functions (black area)

8.5.3 Enlarging identified Time Bounds

Running the anomaly detection in a real plant setup, we encounter the problem of that often a model cannot be identified completely due to the lack of enough input data. The result is a high false positive rate, i.e. in many cases, a correct behavior is classified as anomalous. this is particularly true for the timing behavior. A solution for this problem could be to enlarge the time bounds and therefore accept more of the observed behavior. However, this leads to conflict: The larger the time bounds, which reduce the false positive error rate, the higher is the false negative error rate.

During the anomaly detection phase, the running plant's timing behavior is compared with the prognosis of the identified model. A timing anomaly is signaled whenever a measured timing is outside the timing interval δ (see Definition 15) in the identified Timed Automaton. Here, the interval is defined as $[\mu - k \cdot \sigma, \mu + k \cdot \sigma], k \in \mathbf{R}_+$ where μ is the mean value of the corresponding original observations' timings and σ is the standard deviation.

In a first experiment, the Lemgo Model Factory is used again. A frequently occurring error for example is the wear of a conveyer belt which leads to a decrease in the system's throughput. 12 production cycles are used to identify a normal behavior model. The PTA comprises 6221 states. BUTLA reduces this to 13 states—this corresponds to a compression rate of 99.79%.

To verify the model learning algorithm with a high amount of data, in a second experiment, data is generated artificially using the modified Reber grammar (extended

with timing information). 1000 samples are generated to learn the model, then 2000 test samples are created where 1000 comprise timing errors. From the initial 5377 states in the PTA, a model with 6 states is learned.

Table 8.2 shows the error rates for the anomaly detection applied to both data sets using different factors k.

Table 8.2 Experimental results using real and artificial data.

	k =1	k =2	k =3	k =4
false negative rate (%) - LMF	2	5.3	12.8	30
false positive rate (%) - LMF	12	4.2	2	0
false negative rate (%) - Reber grammar	0	1.3	7.5	21
false positive rate (%) - Reber grammar	9	3.1	1.1	0

The experimental results in Table 8.2 show that the false positive rate could be reduced by enlarging the time bounds. But at the same time, the false negative rate rose. The application of the enlargement of the time requires a tradeoff between false positive and false negative rate. This has to be done separately for each application.

8.6 BUTLA vs. OTALA

Finally, BUTLA and OTALA are compared from the practical point of view.

For this evaluation, a part of the smart factory has been used. It comprises 10 IOs (6 inputs, 4 outputs). The acquired data set comprised 12 production cycles. Since this data set is not sufficient to converge to the correct model, it was extended by simulation as described in Section 8.1.1.3. The final data set comprised 5000 production cycles. However, an expert of the plant guessed that 200 production cycles should be enough to identify a correct behavior model. To keep the scenario realistic, only the first 200 samples were used by BUTLA to learn a model. OTALA needed 500 samples to converge. Both OTALA and BUTLA identified a model with 15 states.

Then, 3000 positive and 100 negative samples where used for anomaly detection. The result is given in the confusion matrix (according to [TSK06]) in Table 8.3. The accuracy is calculated according to Equation 8.3.

Table 8.3 Confusion matrix for experiment in the smart factory.

	true positive f_{TP}	true negative f_{TN}	false positive f_{FP}	false negative f_{FN}	Accuracy
OTALA	100%	98%	2%	0%	99%
BUTLA	100%	76%	24%	0%	88%

It can be seen, that all inserted errors could be found with both identified models (100% true positives and 0% false negative). However, since BUTLA used too little learning examples, the value for false positives is very high (24%). This clarifies

the problem, which is not known in advance that how many samples are needed for learning. The number of false positives for OTALA is smaller (2%) since the learning converged and only some outliers have not been used for learning.

In this experiment, we used only 200 samples for learning the model with BUTLA, because this number was guessed by an expert. This amount of data was not sufficient to converge, and therefore, the accuracy is worse than using OTALA (which learned until convergence). However, using the same number of learning samples for BUTLA, it reaches the same accuracy as OTALA.

It can be seen that both algorithms produce good results, but it reveals the weakness of BUTLA: It is not possible to determine how many learning samples are required to learn a correct model, whereas OTALA is able to recognize convergence autonomously and therefore uses the minimum number of learning samples only. Therefore, OTALA is especially suited for the case if only little knowledge about the process is available.

In Section 9.2, BUTLA and OTALA are compared concerning the quality for anomaly detection using data sets from a real production plant.

8.7 BUTLA vs. alternative methods

The quality of anomaly detection with models identified by BUTLA has also been compared with the quality of anomaly detection with models identified by alternatives methods, namely neural networks and decision tree learning. For this, two subsystems of the LMF have been used (transport modules). First, a learning data set has been used to identify a model (the same data set for each method). Then, a test data set has been used for anomaly detection (again the same data set for each method). The test data set included positive as well as negative examples. So, the results are given in the confusion matrix in Table 8.4.

Table 8.4 Confusion matrix for the experiment in the LMF.

	true positive	true negative	Accuracy
BUTLA	100%	97%	98,5%
Neural Networks	40.2%	42.2%	41.2%
Decision Trees	90.5%	83.3%	86.9%

BUTLA was able to detect all inserted anomalies. The number of false positives is also a good value, only some outliers (mainly timing deviations) have not been learned. In total, a high accuracy of 98.5% is achieved. In contrast, the neural networks and decision trees achieved an accuracy of only 41% and 87% respectively.

Part III

Applications

Chapter 9

Applications

A drawback of many contributions is the lack of real data. Many algorithms are tested on artificial data only, which in many cases are specially tailored for the proposed identification algorithm. These algorithms often fail when they use real data. The algorithms proposed in this thesis are not only used for artificial data, but also in real applications of industrial production plants.

The usability of the identified models for anomaly detection has been shown in various application scenarios. First, the model factory at the Institute Industrial IT in Lemgo has been used to test the basic functionality of the algorithm. The results and some figures are shown in section 9.2. After this, the algorithm has been tested at a real plant at Jowat AG in Detmold (see section 9.3). Due to the size and complexity, this plant was more challenging than the model factory.

The main purpose of the learned models is the anomaly detection in production plants. Therefore, this aspect is specially considered.

The contributions of this chapter are the following:

- We describe the framework proKNOWS, which is an implemented tool chain for model identification, diagnosis and optimization.
- We show the applicability of the proposed identification algorithms for the Lemgo Model Factory, an experimental plant setup.
- We show the applicability of the proposed identification algorithms for a real process plant at Jowat AG.

This Section is organized as follows: First in Section 9.1, the tool box proKNOWS is introduced, which was developed as framework for model identification and diagnosis algorithms. Section 9.2 shows the applicability at the Lemgo Model Factory (LMF), which is an experimental production plant serving for tests of different research work. Finally in Section 9.3, we show the application at a process plant at Jowat AG.

9.1 Tool Box Implementation

In a joint research work with the research institute Fraunhofer IOSB-INA, *ProKNOWS*, a framework for model identification, diagnosis and optimization has been

developed. ProKNOWS comprises several algorithms for the identification of behavior models and for the diagnosis of production plants. Here, among others, BUTLA and OTALA are used for model identification and ANODA for the anomaly detection. Figure 9.1 shows the architecture of the tool box *proKNOWS*.

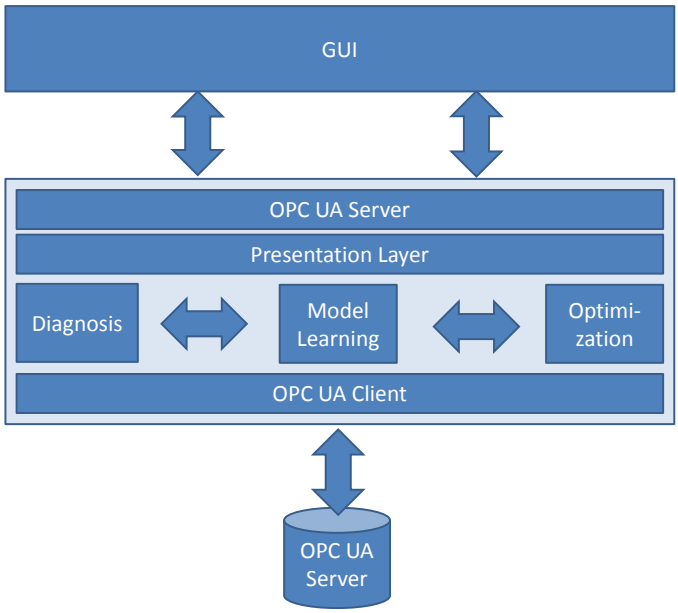


Fig. 9.1 The architecture of the tool box *proKNOWS*.

Using a datalogger as proposed in [PKN+12, PN12], the data is acquired from the production plant and passed to an OPC UA server. Any OPC UA client can connect to this server and subscribe the required signals.

proKNOWS is also equipped with an OPC UA client. The obtained data is forwarded to one of the chosen applications. These are diagnosis, model identification and optimization.

For each module which has to be monitored, a separate instance is created. Here, the BUTLA or OTALA algorithm is used to identify a behavior model for each module. If OTALA is chosen for the identification, the data are received online directly via the OPC UA client. For the identification with BUTLA, the data are stored into a database, before they are used offline to identify a model.

In the diagnosis phase, as for the identification with OTALA, the data are usually received online directly via the OPC UA client. It also provides the possibility to run the anomaly detection on historical data, which is stored in the data base.

Finally, the user is informed about all the actions in a separate graphical user interface (GUI). For this, an OPC UA server-client relationship is established again. The advantage is the platform independence, as any kind of platform can be used for visualization, e.g. iOS or Android devices.

Figure 9.2 shows a screen shot of the graphical user interface in proKnows.

The visualization (in this configuration) consists of four parts:

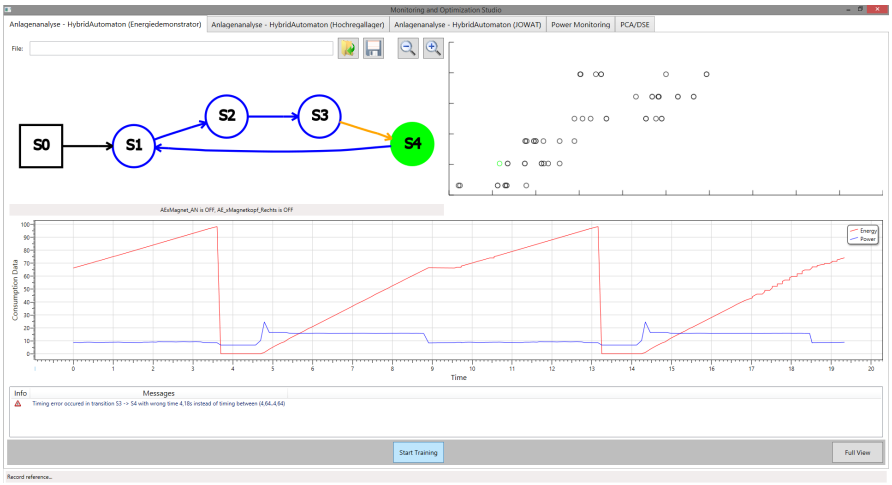


Fig. 9.2 Graphical user interface in proKnows.

- **Top left:** At the top left, the identified automaton is displayed. In the displayed screen shot, currently the diagnosis phase is active. Below the visualized automaton, a detailed information about the last event is given. The current state is colored green, the historical path of the automaton is colored blue. In case of an occurring anomaly, the corresponding elements are colored in a different color to highlight the anomalous behavior. In this example, the transition between state S3 and S4 is marked in a different color because the underlying timing was anomalous.
- **Bottom:** Detailed messages about any possible error are given in the message box at the bottom. In this screen shot, it gives detailed information about the error, which is highlighted in the automaton: A timing error occurred between state S3 and S4.
- **Top right:** In this field some other visualization can be placed. The given screen shot shows the visualization of the Discrete State Encoding (DSE, see [MTN12] for more details).
- **Middle:** The field in the middle allows the visualization of continuous signals. In this example, the visualization of the actual power and energy consumption was chosen.

The framework proKNOWS will continuously be extended by including more identification and diagnosis algorithms.

9.2 Lemgo Model Factory

The Lemgo Model Factory (LMF) is an exemplary plant at the Institute Industrial IT in Lemgo, Germany. It serves as an experimental platform for the first implementations of the research work done at the institute. The LMF consists of eight modules with different roles such as transporting, weighing, packing and storing bulk material (corn) and finally for producing (popcorn). The modules 1, 2 and 3 are used to transport and weigh the bulk material, module 6 is used to pack portions of

corn into bottles, which are transported via a conveyer belt (module 7) to a storing robot (module 8). Finally, in module 4 and 5, popcorn is made from the corn. In total, around 250 sensors and actuators are used, which are controlled by a central PLC and connected via an industrial bus system, Profinet. Figure 9.3 shows the LMF with the SCADA system.

Fig. 9.3 Lemgo Model Factory.

We performed several experiments with some acquired data. Here, we give some results of experiments, using module 1 and 2. In total, 34 examples of the good behavior were available.

Figure 9.4 shows an identified Timed Automaton from the second module in the Lemgo Model Factory. To keep the figure clear, the timing information is not displayed.

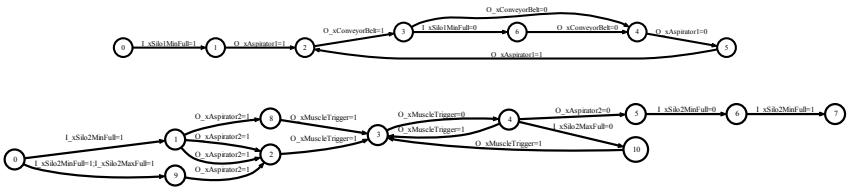


Fig. 9.4 Identified Timed Automata from the first two modules in the Lemgo Model Factory.

First, we performed the 2-fold cross validation to evaluate the correctness and generalization of the model identification algorithms. Since the single examples are independent and the order is arbitrary, we divided the training set into the following groups: *(1)* 1-17 and *(2)* 18-34.

Table 9.1 shows the results for the 2-fold cross validation for LMF data sets of modules 1 and 2. Obviously, the amount of data (17 production cycles) is not sufficient to identify a correct model. BUTLA reaches an average false positive rate of 47%, OTALA even reaches an average rate of 79%.

Table 9.1 Results for 2-fold cross validation for LMF data sets of modules 1 and 2.

Identification algorithm	data set	training subset	test subset	False Positive	average False Positive
BUTLA	LMF 1	1	2	35.3%	47%
		2	1	76.5%	
	LMF 2	1	2	11.8%	
		2	1	64.7%	
OTALA	LMF 1	1	2	76.5%	79.4%
		2	1	82.4%	
	LMF 2	1	2	70.6%	
		2	1	88.2%	

Additionally, a Leave-One-Out cross validation (LOOCV), which is a special variant of the k-fold cross validation, was performed. The results are given in Table 9.2.

Table 9.2 Results for Leave-One-Out cross validation for LMF data sets of modules 1 and 2.

identification algorithm	data set	False Positive	Average False Positive
BUTLA	LMF 1	79.4%	75%
	LMF 2	70.6%	
OTALA	LMF 1	85.3%	83.9%
	LMF 2	82.4%	

Since more data (33 production cycles) were used to identify the models in this experiment, better false positive rates were achieved. BUTLA has a false positive rate of 75% and OTALA has 84%. Also, the difference between the two values decreased. Using more examples for the identification, the difference between the two values would further decrease and finally reach some value close to 100%.

We also performed some experiments with real data and inserted errors. Since errors appear randomly, we used the LMF to insert some anomalous behavior. However, it is difficult to insert errors in a running production plant and create a huge amount of observations. Therefore, only few data are available for the anomaly detection experiment with real errors.

In an experiment we inserted 17 different failures. Using the ANODA algorithm and the identified Timed Automata, we were able to detect 88% of the failures correctly. In the remaining 12% we were able to detect the error, but the error cause was not identified correctly.

Although we were able to detect most of the errors (at least the failures which were enforced by ourselves), we encountered a problem: Often, a correct behavior was recognized as an error. This happens because we are not able to learn the completely correct behavior model. For this, we would need an infinite number of recorded test samples. To prevent this, it is possible to enrich the recorded observations e.g. by using a normal distribution and create additional samples. Another possibility is to adapt the model during runtime. For this, we would need a supervised learning algorithm which allows the plant operator to add a path to the model. This issue is not yet completely solved and should be addressed in future work. Attempts at a solution are given in Section 6.5.

9.3 Plant at Jowat AG

The proposed algorithms have also been tested at a real production plant at the Jowat AG.

The Jowat AG, with headquarters in Detmold, is one of the leading suppliers of industrial adhesives. These are mainly used for woodworking and furniture manufacturing, in the paper and packaging industry, in the textile industry, for 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).

We compared some key values of the identification using BUTLA and two other algorithms from the state of the art: Alergia and MDI. In the experiment, the number of states, comparisons, merges and determinizations have been compared. Since OTALA is an online learning algorithm, not using the state merging approach, it does not fit into this experiment. The results of the algorithms' comparison are given in Table 9.3.

Table 9.3 Algorithm comparison for Jowat AG data.

	Alergia	MDI	BUTLA
#states in PTA	445	445	490
#states in automaton	27	16	13
#merges	418	429	477
#comparisons	1025	605	3702
#determinizations	348	578	188
reduction(%)	93.93	96.4	97.4

The PTA size of both MDI and Alergia are equal. Since BUTLA additionally considers time and splits transitions with multiple time modes, BUTLA creates a bigger PTA. High number of merges and high size reductions are evident for all algorithms. BUTLA does more comparisons. This is due to comparing the timing of transitions, in addition to comparing their probabilities. However, ALERGIA and MDI perform more determinizations since they both proceed top-down and therefore they have to recursively merge the subtrees of compatible states. An expert of the plant has examined the outcome and stated that he was able to see the recipe of the product in the automaton identified by BUTLA.

Due to the secrecy agreement, no figures of the identified automata can be shown here.

The data we obtained included measurements of the correct production operation, without observed erroneous behavior. Since the plant is daily used for production, it

was not possible to insert some errors and to test the anomaly detection capabilities. In future work, the algorithms will be connected to the plant prototypically. The data will be received in real time, such that ANODA can find anomalies during the running production.

Part IV

Conclusion

Chapter 10

Conclusions and Future Work

During the research for this thesis, some questions were answered and some new questions arose. In this chapter, we give a summary of the research work done in this thesis and an outlook to the future work.

10.1 Conclusions

This thesis started with the claim that manual model generation is time and cost intensive. It is desirable to identify models automatically. The target application domain is the automation industry, and production systems naturally depend on time. Therefore in Chapter 2, an appropriate modeling formalism was identified, which is suited to capture timed behavior. Further requirements are the suitability for automatic identification and for anomaly detection. These requirements are met by a Timed Automaton, which additionally scores in the intuitive interpretability and the ability for visualization.

Based on the formalisms that were given in Section 3, in Section 4 the state of the art research of the complexity of identification of finite automata was reviewed. Section 5 summarized some state of the art identification algorithms for finite automata.

In Part II, we presented the main contribution of this thesis. Chapter 6 presented the following algorithms:

- We introduced BUTLA for the identification of Timed Automata. This algorithm firstly uses the bottom-up merging strategy, which results in a computation speed increase.
- We introduced OTALA, which is the first passive online algorithm for the identification of Timed Automata. It is specially designed for the identification of normal behavior in cyber-physical production systems where observations cannot be stored and it is not possible to use expert knowledge.
- We introduced ANODA, the anomaly detection algorithm. It uses the Timed Automata identified by BUTLA or OTALA and finds anomalies in the current plant behavior.

In Chapter 7, these algorithms were analyzed theoretically. The main theorems which we proved are the following:

- BUTLA is weakly polynomially identifiable in the limit.
- BUTLA is strongly polynomially identifiable in the limit under certain conditions.
- BUTLA identifies the timing in the limit.
- The usage of a timing preprocessing instead of the splitting operation leads to a computation speed increase.
- The bottom-up merging strategy runs faster for relevant cases than the top-down merging strategy.
- The anomaly detection problem can be solved in logarithmic space, $ANODA \in \mathcal{L}$.

Furthermore, the proposed algorithms were evaluated empirically. The results are given in Chapter 8. As a main part of that chapter, BUTLA and OTALA are compared concerning characteristic features such as runtime behavior, convergence of identification and ability for anomaly detection.

Finally in Chapter 9, we gave three examples of how the identification and anomaly detection algorithms are used in practice.

- The model identification and diagnosis framework proKNOWS that uses BUTLA and OTALA for model identification and ANODA for anomaly detection.
- The developed algorithms have been used in the LMF, the model factory at the institute. The applicability has been proven by empirical evaluations.
- The model identification algorithms have further been tested in a real process plant at Jowat AG.

These examples prove the applicability of the proposed algorithms for real application scenarios.

10.2 Future Work

Much research work has been done in this thesis. However, not all question have been answered and additional questions arose. These are subject to future work.

Adaptive learning
In section 6.5 we introduced two approaches for the adaptive modification of Timed Automata, which have been identified using BUTLA and OTALA respectively. However, they partially use a work around to adapt the model. Especially for BUTLA, it is still an open question that how the model can be adapted online while the identification algorithm works offline.

Identification of Hybrid Automata
BUTLA has already been extended to HyBUTLA [Vod13] for the identification of hybrid automata. In the next step, OTALA will be extended for the identification of hybrid automata. Here, compared to HyBUTLA, additional research questions are open, for instance, how the continuous signals can be approximated online.

Model-based diagnosis
ANODA, so far, is able to detect anomalies in the behavior of timed systems. Combining with the approach from [Vod13], anomalies in continuous signals

can also be detected. However, model-based diagnosis additionally includes the identification of the error cause, which is still not realized and remains an open topic for future work.

Practical usage of the identified automata in real applications

In Chapter 9 we gave some examples for the practical usage of the identified automata. However, the framework proKNOWS is still a prototypical software and has to be developed further to allow a continuous usage in an industrial environment.

In Sections 9.2 and 9.3 we gave two examples of real plants in which the identified models have been used. However, the usage of the Jowat data is restricted to offline identification so far; this is due to the missing online data connection. Furthermore, at least for the plant in Jowat, no error samples were available. In future work, the proKNOWS will be be connected directly to the plant such that an online analysis can be performed.

List of Figures

List of Tables

Abbreviations

1-DTA	One-Clock Deterministic Timed Automaton
ANODA	Anomaly Detection Algorithm
BUTLA	Bottom-Up Timing Learning Algorithm
CDF	Cumulated Density Function
CTMC_L	Edge Labeled Continuous-time Markov Chains
CYK	Cocke Younger and Kasami
DES	Discrete Event System
DFA	Deterministic Finite Automaton
ERA	Event-Recording Automaton
FN	False Negative
FP	False Positive
FSA	Finite State Automaton
FSM	Finite State Machine
GUI	Graphical User Interface
HIL	Hardware in the loop
HMM	Hidden Marcov Models
HyBUTLA	Hybrid Bottom-Up Timing Learning Algorithm
ID_1DTA	Identification of 1-clock Deterministic Timed Automata
IO	Input/Output
LMF	Lemgo Model Factory
LOOCV	Leave One Out Cross Validation
MBD	Model-Based Diagnosis
MDI	Minimal Divergence Inference
n-TA	n-clock Deterministic Timed Automaton
NDAOO	Non-Deterministic Autonomous Automaton with Output
NFA	Nondeterministic Finite Automata
OTALA	Online Timed Automaton Learning Algorithm
PAC	Probably Approximately Correct
PDF	Probability Density Function
PDFA	Probabilistic Deterministic Finite Automaton
PDRTA	Probabilistic Deterministic Real-Time Automaton
PDTA	Probabilistic Deterministic Timed Automaton
PHyTA	Probabilistic Deterministic Hybrid Timed Automaton
PLC	Programmable Logical Controller
PTA	Prefix Tree Acceptor
RPNI	Regular Positive and Negative Inference
RTI	Real-Time Identification
RTI+	Real-Time Identification from Positive Data
SCADA	Supervisory Control and Data Acquisition

SFA	Stochastic Finite Automaton
TA	Timed Automaton
TN	True Negative
TP	True Positive

References

[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*, 138:3–34, 1995.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, vol. 126:183–235, 1994.

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211:1–13, 1999.

[AH92] Rajeev Alur and Thomas A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *In Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 177–186. IEEE Computer Society Press, 1992.

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comp.*, pages 75(2):87–106, 1987.

[Aut14] AutomationML. www.automationml.org, 2014.

[Bar10] A.M. Bartkowiak. Anomaly, novelty, one-class classification: A short introduction. In *Computer Information Systems and Industrial Management Applications (CISIM), 2010 International Conference on*, 2010.

[BBN93] Mich Ele Basseville, Michèle Basseville, and Igor V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, 1993.

[BC11] Alberto Bemporad and Stefano Di Cairano. Model-predictive control of discrete hybrid stochastic automata. *IEEE Trans. Automat. Contr.*, 56(6):1307–1321, 2011.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[BGK10] Z I Botev, J F Grotowski, and D P Kroese. Kernel density estimation via diffusion. *Annals of Statistics*, 38(5):2916–2957, 2010.

[BJ94] R. Benjamins and W. Jansweijer. Toward a competence theory of diagnosis. *IEEE Expert*, 9(5):43–52, Oct 1994.

[CCW+11] Ming Chang, Hongwei Chen, Yingying Wang, Yueou Ren, and Hao Kang. The application of neural network in the intelligent fault diagnosis system. In *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*, pages 412–415, Dec 2011.

[Cer93] A. Cerone. *A Net-based Approach for Specifying Real-time Systems*. Serie TD. Ed. ETS, 1993.

[CG08] Jorge Castro and Ricard Gavaldà. Towards feasible pac-learning of probabilistic deterministic finite automata. In *Proceedings of the 9th international colloquium on Grammatical Inference: Algorithms and Applications*, ICGI '08, pages 163–174, Berlin, Heidelberg, 2008. Springer-Verlag.

[CGP99] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[CGS07] M.P. Cabasino, A. Giua, and C. Seatzu. Identification of Petri Nets from Knowledge of Their Language. *Discrete Event Dynamic Systems*, 17(4):447–474, 2007.

[CL06] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[CMS99] Antonio Cerone and Andrea Maggiolo-Schettini. Time-based expressivity of time petri nets for system specification. *Theoretical Computer Science*, 216(1 - 2):1 – 53, 1999.

[CO94] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In *GRAMMATICAL INFERENCE AND APPLICATIONS*, pages 139–152. Springer-Verlag, 1994.

[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.

[CPD11] Luca Console, Claudia Picardi, and Daniele Theseider Dupré. Temporal decision trees: Model-based diagnosis of dynamic systems on-board. *CoRR*, abs/1106.5268, 2011.

[CT04] Alexander Clark and Franck Thollard. Pac-learnability of probabilistic deterministic finite state automata. *J. Mach. Learn. Res.*, 5:473–497, December 2004.

[DLH97] Colin De La Higuera. Characteristic sets for polynomial grammatical inference. *Mach. Learn.*, 27(2):125–138, May 1997.

[dlH05] Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recogn.*, 38(9):1332–1348, September 2005.

[dlH10] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, 2010.

[DLT04] F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using rfsas. *Computer Science*, pages 313(2):267–294, 2004.

[Dom99] Pedro Domingos. The role of Occam's Razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3:409–425, 1999.

[EDJ+71] W. T. Eadie, D. Drijard, F. E. James, M. Roos, and B. Sadoulet. *Statistical Methods in Experimental Physics*. North Holland, 1971.

[FMMR10] Carlo A. Furia, Dino Mandrioli, Angelo Morzenti, and Matteo Rossi. Modeling time in computing: A taxonomy and a comparative survey. *ACM Comput. Surv.*, 42(2):6:1–6:59, March 2010.

[Fre08] Christian W. Frey. Diagnosis and monitoring of complex industrial processes based on self-organizing maps and watershed transformations. In *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, 2008.

[GJL04] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. Learning of event-recording automata. In *In 6th International Workshop on Verification of Infinite-State Systems, volume 138/4 of Electronic Notes in Theoretical Computer Science*, pages 379–395. Springer, 2004.

[GKN+11] Olaf Graeser, Barath Kumar, Oliver Niggemann, Natalia Moriz, and Alexander Maier. Automationml as a basis for offline- and realtime-simulation. In *8th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2011)*, July 2011.

[GLL99] A. Girault, B. Lee, and E.A. Lee. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-Aided Design*, 18(6):742–760, June 1999. Research report UCB/ERL M97/57.

[Gol67] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447 – 474, 1967.

[Gol78] E Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302 – 320, 1978.

[Hen02] M.M. Henry. *Model-based Estimation of Probabilistic Hybrid Automata*. Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 2002.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):pp. 13–30, 1963.

[HT00] Colin Higuera and Franck Thollard. Identification in the limit with probability one of stochastic deterministic finite automata. In ArlindoL. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Computer Science*, pages 141–156. Springer Berlin Heidelberg, 2000.

[HW02] Michael W. Hofbaur and Brian C. Williams. Mode estimation of probabilistic hybrid systems. In *In Intl. Conf. on Hybrid Systems: Computation and Control*, pages 253–266. Springer Verlag, 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, July 2003.

[IM10] R. Isermann and M. Münchhof. *Identification of Dynamic Systems: An Introduction with Applications*. Advanced textbooks in control and signal processing. Springer, 2010.

[Ise04] Rolf Isermann. Model-based fault detection and diagnosis - status and applications. In *16th IFAC Symposium on Automatic Control in Aerospace*, St. Petersbug, Russia, 2004.

[JJN11] Michael Jaeger, Roman Just, and Oliver Niggemann. Using automatic topology discovery to diagnose profinet networks. In *16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011)*, Toulouse, France, Sep 2011.

[JKH08] M. Jager, C. Knoll, and F.A. Hamprecht. Weakly supervised learning of a classifier for unusual event detection. *Image Processing, IEEE Transactions on*, 17:1700–1708, 2008.

[KAB14] T. Klerx, M. Anderka, and H. Kleine Büning. On the usage of behavior models to detect atm fraud. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 2014.

[KDSJ14] Barath Kumar, Rainer Drath, Ben Schroeder, and Jürgen Jasperneite. Test generation from functional 3d virtual environment models. In *ETSI User Conference on Advanced Automated Testing (UCAAT 2014)*, Munich, Sep 2014.

[Kha02] Hassan Khalil. *Nonlinear Systems*. Prentice Hall, January 2002.

[KNJ10] Barath Kumar, Oliver Niggemann, and Jürgen Jasperneite. Statistical models of network traffic. In *International Conference on Computer, Electrical and Systems Science*. Cape Town, South Africa, Jan 2010.

[KNSJ12] Barath Kumar, Oliver Niggemann, Wilhelm Schäfer, and Jürgen Jasperneite. Modeling and testing of automation systems. *Advances in Intelligent and Soft Computing*, 133:1027–1034, Mar 2012.

[Kol63] A.N. Kolmogorov. On tables of random numbers. *Sankhya*, Ser. A.(25):369 – 375, 1963.

[Kol98] A.N. Kolmogorov. On tables of random numbers. *Theoretical Computer Science*, 207(2):387 – 395, 1998.

[Koy91] Ron Koymans. (real) time: A philosophical perspective. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 1991.

[KSSN14] Björn Kroll, David Schaffranek, Sebastian Schriegel, and Oliver Niggemann. System modeling based on machine learning for anomaly detection and predictive maintenance in industrial plants. In *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Barcelona, Sep 2014.

[KTN+12] Barath Kumar, Andreas Toensfeuerborn, Oliver Niggemann, Wilhelm Schäfer, and Jürgen Jasperneite. Experience in deploying mbt for industrial automation. In *ETSI Model Based Testing User Conference (MBTUC) 2012.*, Tallinn, Estonia, Sep 2012. ETSI.

[LAD+11] Shang-Wei Lin, Étienne André, JinSong Dong, Jun Sun, and Yang Liu. An efficient algorithm for learning event-recording automata. In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis*, volume 6996 of *Lecture Notes in Computer Science*, pages 463–472. Springer Berlin Heidelberg, 2011.

[LH11] W. Liu and I. Hwang. Robust estimation and fault detection and isolation algorithms for stochastic linear hybrid systems with unknown fault input. *Control Theory Applications, IET*, 5(12):1353 –1368, august 2011.

[LL98] Bilung Lee and Edward A. Lee. Hierarchical concurrent finite state machines in ptolemy. In *ACSD*, pages 34–40. IEEE Computer Society, 1998.

[LPP98] Kevin Lang, Barak Pearlmutter, and Rodney Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm, 1998.

[LSS01] J. Lunze, J. Schröder, and P. Supavatanakul. Diagnosis of discrete event systems: the method and an example. In *Proceedings of the Workshop on Principles of Diagnosis, DX'01*, pages 111–118, Via Lattea, Italy, 2001.

[Mai14] Alexander Maier. Online passive learning of timed automata for cyber-physical production systems. In *The 12th IEEE International Conference on Industrial Informatics (INDIN 2014)*. Porto Alegre, Brazil, Jul 2014.

[MF76] P.M. Merlin and David J. Farber. Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036–1043, Sep 1976.

[MKPGN13] Alexander Maier, Markus Koester, Carlos Paiz Gatica, and Oliver Niggemann. Automated generation of timing models in distributed production plants. In *IEEE International Conference on Industrial Technology (ICIT 2013), Cape Town, South Africa*, Feb 2013.

[MNV+11] Alexander Maier, Oliver Niggemann, Asmir Vodenčarević, Roman Just, and Michael Jaeger. Anomaly detection in production plants using timed automata. In *8th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*. Noordwijkerhout, The Netherlands, Jul 2011.

[MS03a] M. Markou and S. Singh. Novelty detection: A review - part 1. Department of Computer Science, PANN Research, University of Exeter, United Kingdom, 2003.

[MS03b] M. Markou and S. Singh. Novelty detection: A review - part 2. Department of Computer Science, PANN Research, University of Exeter, United Kingdom, 2003.

[MTN12] Alexander Maier, Tim Tack, and Oliver Niggemann. Visual anomaly detection in production plants. In *9th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*. Rome, Italy, Jul 2012.

[MWS13] M.M. Mansour, Mohamed A.A. Wahab, and Wael M. Soliman. Petri nets for fault diagnosis of large power generation station. *Ain Shams Engineering Journal*, 4(4):831 – 842, 2013.

[MWvdBD03] Laura Maruster, A. Weijters, Antal van den Bosch, and Walter Daelemans. Discovering process models by rule set induction. In *Proceedings of the 5th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing*, 2003.

[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.

[NDZ13] Payam Nazemzadeh, Abbas Dideban, and Meisam Zareiee. Fault modeling in discrete event systems using petri nets. *ACM Trans. Embed. Comput. Syst.*, 12(1):12:1–12:19, January 2013.

[NSV+12] Oliver Niggemann, Benno Stein, Asmir Vodenčarević, Alexander Maier, and Hans 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.

[NVM+13] Oliver Niggemann, Asmir Vodencarevic, Alexander Maier, Stefan Windmann, and Hans Kleine Büning. A learning anomaly detection algorithm for hybrid manufacturing systems. In *The 24th International Workshop on Principles of Diagnosis (DX-2013)*, Jerusalem, Israel, Oct 2013.

[OG92] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Machine Perception and Artificial Intelligence*. World Scientific, 1992.

[OHN14] Jens Otto, Steffen Henning, and Oliver Niggemann. Why cyber-physical production systems need a descriptive engineering approach - a case study in plug produce. In *2nd International Conference on System-integrated Intelligence (SysInt)*, Bremen, Germany, Jul 2014.

[Pea05] Karl Pearson. The problem of the random walk. *Nature 72*, pages 294–294, 1905.

[Pet62] C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, pages 386–390, 1962.

[Pit89] L. Pitt. Inductive Inference, DFAs, and Computational Complexity. In K.P. Jantke, editor, *Proceedings of International Workshop on Analogical and Inductive Inference (AII)*, volume 397 of *Lecture Notes in Computer Science*, pages 18–44. Springer-Verlag, Berlin, 1989.

[PKN+12] Florian Pethig, Bjoern Kroll, Oliver Niggemann, Alexander Maier, and Tim Tack. A generic synchronized data acquisition solution for distributed automation systems.

In *17th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2012)*, Krakow, Poland, Sep 2012.

[PN12] Florian Pethig and Oliver Niggemann. A process data acquisition architecture for distributed industrial networks. In *Embedded World Conference 2012*, Mar 2012.

[PW93] Leonard Pitt and Manfred K. Warmuth. The minimum consistent dfa problem cannot be approximated within any polynomial. *J. ACM*, 40(1):95–142, January 1993.

[Qui90] J. R. Quinlan. Induction of decision trees. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[Reb67] A. S. Reber. Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior*, 6:855–863, 1967.

[RLL10] M. Roth, J. Lesage, and L. Litz. Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems. In *American Control Conference (ACC), 2010*, pages 2601–2606, June 2010.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[Rot10] Matthias Roth. *Identification and Fault Diagnosis of Industrial Closed-loop Discrete Event Systems: Identifikation und Fehlerdiagnose Industrieller Ereignisdiskreter Closed-Loop-Systeme*. Logos-Verlag, 2010.

[RSLL12] Matthias Roth, Stefan Schneider, Jean-Jacques Lesage, and Lothar Litz. Fault detection and isolation in manufacturing systems with an identified discrete event model. *Int. J. Systems Science*, 43(10):1826–1841, 2012.

[RT11] V.R. Ravi and T. Thyagarajan. Application of hybrid modeling techniques to a non linear hybrid system. In *Process Automation, Control and Computing (PACC), 2011 International Conference on*, pages 1 –7, July 2011.

[SADMK07] Zineb Simeu-Abazi, Maria Di Mascolo, and Michal Knotek. Diagnosis of discrete event systems using timed automata. In *International Conference on cost effective automation in Networked Product Development and Manufacturing*, Monterrey, Mexico, 2007.

[SCZ+09] Aishe Shui, Weimin Chen, Peng Zhang, Shunren Hu, and Xiaowei Huang. Review of fault diagnosis in control systems. In *Control and Decision Conference, 2009. CCDC '09. Chinese*, pages 5324 –5329, June 2009.

[SFJL03] P. Supavatanakul, C. Falkenberg, and J. J. Lunze. Identification of timed discrete-event models for diagnosis, 2003.

[SH09] C.E. Seah and Inseok Hwang. Fault detection and isolation for stochastic linear hybrid systems. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 2622 –2627, Dez 2009.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–, july, october 1948.

[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.

[SMK+13] Nikolai Schetinin, Natalia Moriz, Barath Kumar, Sebastian Faltinski, Oliver Niggemann, and Alexander Maier. Why do verification approaches in automation rarely use hil-test? In *International Conference on Industrial Technology (ICIT) 25.-27. February 2013, Cape Town, South Africa,*, Feb 2013.

[Sol64] R.J. Solomonoff. A formal theory of inductive inference. part i. *Information and Control*, 7(1):1 – 22, 1964.

[SS80] J. Segen and A. Sanderson. Detecting change in a time-series (corresp.). *Information Theory, IEEE Transactions on*, 26(2):249 – 254, Mar 1980.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, Wien, New York, 1973.

[SVA04] Koushik Sen, M. Viswanathan, and Gul Agha. Learning continuous time markov chains from sample executions. In *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, pages 146–155, Sept 2004.

[TB73] B.A. Trakhtenbrot and J.M. Barzdin. North-Holland, 1973.

[TDdlH00] Franck Thollard, Pierre Dupont, and Colin de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proc. of the 17th International Conf. on Machine Learning*, pages 975–982. Morgan Kaufmann, 2000.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, pages 133–191. MIT Press, Cambridge, MA, USA, 1990.

[Ton01] Simon Tong. *Active Learning: Theory and Applications*. PhD thesis, Stanford University, Stanford, CA, USA, 2001.

[Tri02] Stavros Tripakis. Fault diagnosis for timed automata. In Werner Damm and Ernst-Rüdiger Olderog, editors, *FTRTFT*, volume 2469 of *Lecture Notes in Computer Science*, pages 205–224. Springer, 2002.

[TSK06] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.

[Val84] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.

[Var01] Moshe Y. Vardi. Branching vs. linear time: Final showdown. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2001, pages 1–22, London, UK, UK, 2001. Springer-Verlag.

[VBNM11] A. Vodenčarević, H. Kleine Büning, O. Niggemann, and A. Maier. Using behavior models for anomaly detection in hybrid systems. In *Proceedings of the 23rd International Symposium on Information, Communication and Automation Technologies-ICAT 2011*, 2011.

[VdWW06] Sicco E. Verwer, Mathijs M. de Weerdt, and Cees Witteveen. Identifying an automaton model for timed data. In Yvan Saeys, Elena Tsiporkova, Bernard De Baets, and Yves van de Peer, editors, *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn)*, pages 57–64. Benelearn, 2006.

[VdWW07] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. An algorithm for learning real-time automata. In Maarten van Someren, Sophia Katrenko, and Pieter Adriaans, editors, *Proc. of the Sixteenth Annual Machine Learning Conference of Belgium andthe Netherlands (Benelearn)*, pages 128–135, 2007.

[VdWW08] Sicco E. Verwer, Mathijs M. de Weerdt, and Cees Witteveen. Efficiently learning simple timed automata. In Will Bridewell, Toon Calders, Ana Karla de Medeiros, Stefan Kramer, Mykola Pechenizkiy, and Ljupco Todorovski, editors, *Induction of Process Models*, pages 61–68. University of Antwerp, 2008. Workshop at ECML PKDD.

[VdWW09] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. One-clock deterministic timed automata are efficiently identifiable in the limit. In Adrian Horia Dediu, Armand Mihai Ionescu, and Carlos Martin-Vide, editors, *Language and Automata Theory and Applications*, pages 740–751. Springer, 2009.

[Ver10] Sicco Verwer. *Efficient Identification of Timed Automata: Theory and Practice*. PhD thesis, Delft University of Technology, 2010.

[VKBNM11] Asmir Vodenčarević, Hans Kleine Büning, Oliver Niggemann, and Alexander Maier. Identifying behavior models for process plants. In *Proc. of the 16th IEEE International Conf. on Emerging Technologies and Factory Automation ETFA'2011*, pages 937–944, Toulouse, France, September 2011.

[VMN13] Asmir Vodenčarević, Alexander Maier, and Oliver Niggemann. Evaluating learning algorithms for stochastic finite automata. In *2nd International Conference on Pattern Recognition Applications and Methods (ICPRAM 2013); Barcelona, Spain*, Feb 2013.

[Vod13] Asmir Vodenčarević. *Identifying Behavior Models for Hybrid Production Systems*. PhD thesis, University of Paderborn, 2013.

[VWW08] Sicco Verwer, Mathijs Weerdt, and Cees Witteveen. Polynomial distinguishability of timed automata. In Alexander Clark, François Coste, and Laurent Miclet, editors, *Grammatical Inference: Algorithms and Applications*, volume 5278 of *Lecture Notes in Computer Science*, pages 238–251. Springer Berlin Heidelberg, 2008.

[WD09] M. Wang and R. Dearden. Detecting and Learning Unknown Fault States in Hybrid Diagnosis. In *Proceedings of the 20th International Workshop on Principles of Diagnosis, DX09*, pages 19–26, Stockholm, Sweden, 2009.

[ZKH+05] Feng Zhao, Xenofon D. Koutsoukos, Horst W. Haussecker, James Reich, and Patrick Cheung. Monitoring and fault diagnosis of hybrid systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(6):1225–1240, 2005.