Data-Driven Identification of Models for Discrete and Hybrid Systems

Dissertation (monograph) approved
by the Doctoral Degree Committee of
Hamburg University of Technology
in pursuit of the academic degree of

Doktor-Ingenieurin (Dr.-Ing.)

written by
Swantje Plambeck

from
Hamburg

2026

1. Reviewer
Prof. Dr.-Ing. Görschwin Fey

2. Reviewer
Prof. Dr. Louise Travé-Massuyès

Chair of the Examination Committee
Prof. Dr.-Ing. Gerhard Bauch

Date of the Oral Examination
25.02.2026

DOI (Digital Object Identifier)
10.15480/882.17166

 https://orcid.org/0000-0002-4875-5280

Summary

In this thesis, we develop data-driven methods for learning interpretable models of discrete and hybrid Cyber-Physical Systems (CPS). Manual modeling of complex CPS is often infeasible and prone to errors, making automated approaches essential for system understanding, testing, and optimization. This work has four contributions in data-driven model identification for CPS: first, analyzing the applicability of classical finite state machine learning to CPS, second, examining the performance of decision trees for modeling temporal behavior, third, developing methods for learning hybrid automata, and fourth, comparing the proposed approaches on qualitative and quantitative criteria.

Three modeling strategies are presented spanning from discrete to hybrid models. First, the work extends classical automata learning with automata forests to improve robustness under data uncertainty while maintaining interpretability. Second, decision tree learning is specifically extended for CPS. The decision tree model handles non-deterministic behavior, establishing theoretical connections to deterministic finite automata and revealing fundamental limitations of finite-horizon model learning. Third, the work proposes two novel algorithms for hybrid automata learning that integrate discrete and continuous dynamics through symbolic regression. These algorithms enable efficient learning of hybrid systems and leverage system dynamics to identify transition points between modes.

The proposed methods are evaluated on a diverse set of benchmark and real-world systems, demonstrating their effectiveness in capturing complex behaviors while remaining interpretable. We assess and compare the performance of the approaches on qualitative criteria such as interpretability and robustness as well as quantitative metrics including accuracy and computational efficiency. The results show that the approaches form a progression of complementary methods, each addressing different aspects of CPS modeling and providing a solid base for CPS-driven applications. The contributions advance the state-of-the-art in CPS modeling by providing robust, interpretable solutions.

Acknowledgement

I am deeply thankful for everybody who accompanied me on this path. For me, the most valuable part of research was the discussions with colleagues and peers. In this sense, my deepest thanks go to Görschwin, my PhD supervisor. I could not have imagined a better kind of supervision. He always had a next idea when I was missing one, but never pushed me in a direction I did not want to go, supporting my open-minded way of working and even my engagements beyond scientific topics. His detailed comments on papers, presentations, and proposals were invaluable, and I assume that none of my papers would have been accepted without his advice. Whatever document you sent to Görschwin for feedback, he would send it back with dozens of comments — some of them barely readable. You might enter a discussion worried about harsh feedback, often hearing him begin with, "This is already in a very good state." You would then leave with a concrete plan, a lot of motivation, and the feeling that something great was being created.

A great impact and new research impulses came from Louise, my second supervisor. As my host at LAAS CNRS in Toulouse, France, she could not have offered me a more welcoming experience. The research visit allowed me to follow my own ideas even further, and discussions with Louise always gave me extra motivation. She had a remarkable talent for connecting me with the right people and was always reliable in providing feedback on our papers. Her active contribution in the writing process was impressive for all of us. In this scope, I also want to thank Charlotte, Elodie, Ibis, Léonie, Louis, Lucas, Patricia, Pauline, Rafael, Rahma, Silvano, Yannick, Xavier, and everybody else from the LAAS CNRS DISCO Team and related researchers.

Apart from my research visit, another major impulse came when Maxi joined our team. As I mentioned before, scientific discussions were always the most beneficial part of my work, and with Maxi joining, I was no longer alone with my topic. I am greatly thankful for all the discussions we had, still have and for sure will have in the future. I was incredibly lucky that the person joining my topic was so extraordinarily intelligent and talented, yet still kind enough to answer my dumb questions and tolerate my terrible code quality in proof-of-concept implementations and software projects. His strong opinions and fast mind matched perfectly with my optimism and harmonious way of working. In this scope, I also want to thank Ahmad, Elke, Finn, Franziska-Sophie, Gianluca, Lutz, Mohammad, Paula, Sabine, Srinidhi, and everybody else who was or is a member of the CE group at the IES. Further thanks also go to Hendrik, Jakob,

Johannes, Manav, Markus, Philipp, Sean, and everybody from the ITL, who hosted Maxi and me every Tuesday for our project work day. A further contribution were the students working with me on the topic, who were always a great help and with whom I had many interesting discussions. I want to especially mention Aaron and Arne who worked with me as student assistants for many months and significantly contributed to the work in this thesis. A special thanks also to Ulrike, who recently joined our scientific activities, and whose collaboration I am very much looking forward to continuing.

Beyond my scientific work, I am also grateful for everybody who supported me already during my studies and in my "extracurricular" activities. Special thanks go to Jan, who has been a close companion throughout my studies and PhD, always challenging me in many aspects of both academic and personal life, and being my anchor to student life activities. Another such anchor and challenge is Til, who came into my life through our IEEE & VDE Student Group. I wouldn't have finished the writing part of my PhD without his garden office. When asking Til for feedback, you would receive not only the most honest response, but also one of the most significant and valuable. I cannot wait for what the future brings for us. In this scope, I also want to thank Antje, Dimitri, Jan, Lennart, and Roya from the PhD council; Aditya, Feyona, Jony, Kowsalya, Lucas, Nisal, Ole, Sanaz, Sebastian, Sreelakshmi, and Yevhenii from the IEEE & VDE Student Group; and Kathrin, Oke, and Suresh from the language café. All these activities helped me focus on other things from time to time and kept me from getting too stuck in my research. Further, they helped to get through some harder times of rejected papers and slow scientific progress.

I am also very thankful for all the support I received in my personal life. Of course, it would never have been possible for me to start, continue, and finish this PhD without my parents, Andrea and Lutz, who never doubted that I would be able to do this, even when I doubted myself. My brother is, of course, one of the most important people and pillars in my life. Having an older brother helps a lot in getting through school and university — including being able to copy all the best practices. I also want to thank everyone in my family who, while not part of my scientific journey directly, always had my back and was proud of every bit of progress I made.

Finally, an important part in life are friends. While many of the people mentioned above are close friends as well, there are still more to mention. My deepest personal thanks for the pursuit of my PhD go to Paavo. I cannot express in how many ways he supported my life and personal growth, and I would never have made it through this without him. Supporting each other's personal and professional goals strengthened both of us. Apart from that, I want to thank Antje, Christian F., Christian H., Denise, Henk, Marita, and Lukas for getting me out of my office and my PhD bubble from time to time.

Illustrative figures have been partly generated using ChatGPT and some parts of the text have been furnished with the help of Microsoft CoPilot.

Contents

Summary iii
Acknowledgement v
Acronyms xi
Notations and Symbols xv
1 Introduction 1
1.1 Cyber-Physical Systems in the Context of Digitization 1
1.2 Motivating Example for Modeling Challenges in CPS 3
1.2.1 Challenge 1: Identification of Discrete System Models 3
1.2.2 Challenge 2: Integration of Temporal Information 5
1.2.3 Challenge 3: Capturing Continuous & Hybrid Dynamics 6
1.2.4 Challenge 4: Evaluation & Comparison of Modeling Approaches 7
1.2.5 Insights and Research Gaps 9
1.3 Contributions & Structure 9
2 Related Work 15
2.1 Related Fields in Model Identification 15
2.1.1 Digital Twins 16
2.1.2 Abstraction Methods & Abstract Models 17
2.1.3 Manual Modeling 18
2.1.4 Machine Learning 18
2.1.5 Physics-Informed Learning 19
2.1.6 Finite State Machines & Hybrid Automata 20
2.1.7 Probabilistic Models 20
2.2 Learnability - A Systematic Literature Review 21
2.2.1 Methodology 21
2.2.2 Spectrum of the Literature with respect to Data-Driven Modeling 23
2.2.3 Learnability in the Learning Process of Cyber-Physical Systems 25
2.3 Summary 26
3 Preliminaries 27
3.1 Decision Trees 27
3.1.1 Decision Tree Learning 28
3.1.2 Regression Trees 29
3.2 Finite State Machines 29
3.2.1 Passive Automata Learning 32
3.2.2 Active Automata Learning 33

3.3 Hybrid Automata ... 34
3.4 System & Observations ... 35
3.5 Dynamic Time Warping ... 36
3.6 Symbolic Regression ... 37
3.7 Examples ... 39
3.7.1 Coffee Machine ... 41
3.7.2 Water Tank ... 42
3.7.3 Two Tank System ... 43
3.7.4 Boiler ... 44
3.7.5 Power Converter ... 45
3.7.6 Vacuum Cleaner ... 47
4 Identification of Deterministic Finite Automata ... 49
4.1 Related Work ... 51
4.2 System Abstraction ... 52
4.3 Capabilities of Passive & Active Automata Learning for Cyber-Physical Systems ... 55
4.3.1 System Observation & Learning Strategy ... 55
4.3.2 Case Study: Vacuum Cleaner ... 56
4.3.3 Summary & Discussion ... 60
4.4 Test Case Generation with Automata Learning ... 61
4.4.1 Experimental Setup & Data Aggregation ... 62
4.4.2 Automata Learning ... 63
4.4.3 Test Case Generation on the Automaton Model ... 65
4.5 Automata Forests ... 66
4.5.1 Formalization & Learning Algorithms ... 66
4.5.2 Discussion & Properties ... 68
4.5.3 Empirical Results ... 71
4.6 Summary ... 75
5 Decision Tree Models ... 77
5.1 Related Work ... 79
5.2 Discrete Decision Tree Models ... 80
5.2.1 Observations & Decision Tree Learning ... 80
5.2.2 Language-Based Model & Decision Tree Language ... 82
5.2.3 Output-Based Model & Error Bounds ... 85
5.2.4 Prediction Model & Monitoring ... 87
5.2.4.1 Complete Scenario ... 89
5.2.4.2 Practical Scenario ... 90
5.3 Regression Tree Models ... 93
5.3.1 Regression Tree Modeling ... 93
5.3.2 Empirical Evaluation ... 94
5.4 Test Case Generation with Decision Tree Models ... 97
5.4.1 Finite State Machine Representation of the Decision Tree Model ... 97

5.4.2 Leaf Coverage Criterion 99
5.4.3 Automatic Test Generation 100
5.4.4 Case Study 104
5.5 Summary 106
6 Identification of Hybrid Automata 109
6.1 Related Work 111
6.2 Hybrid Decision Tree — Structure, Modeling Process & Evaluation 112
6.3 Fast Model Learning for Hybrid Systems 116
6.3.1 Trace Segmentation 116
6.3.2 Segment Grouping 118
6.3.3 Mode Characterization 122
6.3.4 Model Construction 123
6.3.5 Evaluation 123
6.4 Symbolic Regression for Hybrid Decision Tree Learning 130
6.4.1 Trace Segmentation 131
6.4.2 Segment Grouping & Mode Characterization 134
6.4.3 Model Construction 136
6.4.4 Evaluation 137
6.5 Summary 142
7 Application & Comparison of Learning Approaches 145
7.1 Related Work 147
7.2 Practical & Real-World Applications of Abstract Learned Models 149
7.3 A General Framework for Learning Models in CPS-driven Applications 150
7.4 Model Properties & Capabilities 155
7.5 Model Performance 157
7.5.1 Quantitative Comparison of Discrete Models 159
7.5.2 Quantitative Comparison of Continuous Models 161
7.6 Summary 163
8 Conclusion 167
8.1 Impact 167
8.2 Future Work 169
Bibliography 170
Index of Figures 183
Index of Tables 188
Index of Listings 190

Acronyms

ARX – **AutoRegressive eXogenous** 116, 122, 123, 130, 143, 157

CPS – **Cyber-Physical System** 1, 2, 3, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 35, 41, 47, 49, 50, 51, 52, 55, 56, 60, 65, 66, 75, 76, 77, 78, 79, 97, 106, 107, 109, 110, 142, 143, 146, 147, 148, 149, 150, 151, 152, 155, 159, 163, 164, 165, 167, 168, 169, 183, 186, 188

DFA – **Deterministic Finite Automaton** xviii, 10, 11, 12, 20, 26, 27, 30, 31, 32, 33, 47, 49, 50, 52, 57, 67, 77, 109, 145, 146, 155, 156, 157, 168, 184, 190

DTL – **Decision Tree Learning** 2, 5, 6, 7, 10, 12, 28, 29, 76, 78, 79, 80, 82, 88, 96, 106, 123, 130, 137, 142, 155, 156, 157, 158, 159, 161, 168, 183, 190

DTW – **Dynamic Time Warping** xvii, 27, 36, 37, 118, 119, 120, 121, 122, 127, 143, 184, 187, 191

EDSM – **Elastic-Degenerate String Matching** 51

FSM – **Finite State Machine** xv, xvi, 5, 9, 10, 11, 25, 29, 30, 32, 34, 49, 50, 51, 54, 61, 77, 78, 79, 97, 98, 99, 100, 103, 104, 105, 106, 107, 109, 145, 146, 183, 187

IMU – **Inertial Measurement Unit** 4

LiDAR – **Light Detection and Ranging** 3, 4, 6, 7, 8, 9, 10, 12, 62, 78, 183

LLM – **Large Language Model** 22

LMI – **Linear Matrix Inequality** xviii, 119, 122, 124, 125, 126, 129, 191

MSE – **Mean Squared Error** 29, 38, 95, 115, 124, 127, 140, 141, 157, 189, 190

PAC – **Probably Approximately Correct** 17, 20, 26, 79

PTA – **Prefix Tree Acceptor** 32, 33, 111, 124, 130

RPNI – **Regular Positive-Negative Inference** 32, 33, 51, 54, 55, 68, 72, 73, 74, 75, 79, 155, 189, 190

RTL – **Regression Tree Learning** 6, 7, 8, 29, 155, 156, 157, 158, 162, 183

RTLS – **Real-Time Localization System** 3, 5, 6, 50, 61, 62, 63, 64, 65, 76, 149, 152, 157, 158, 183, 186

***SR* – Symbolic Regression** xv, xvii, xviii, 37, 130, 131, 132, 133, 134, 135, 136, 137, 138, 142, 143, 155, 156, 158, 161, 162, 188, 190, 191

***STL* – Signal Temporal Logic** 17, 25

***SUL* – System Under Learning** 11, 12, 32, 33, 35, 53, 55, 56, 76, 185

Notations and Symbols

Symbol	Usage / Meaning
$\mathbb{A}$	Set of Finite State Machines (FSMs)
$\mathbb{D}$	Categorization Subspace
$\mathbb{F}$	Space or Value Set of a Feature
$\mathbb{N}$	Natural Numbers
$\mathbb{R}$	Real Numbers
$\mathcal{A}$	Automaton / Finite State Machine (FSM)
$\mathcal{F}$	Automata Forest
$\mathcal{O}$	Set of Observations
$\mathcal{S}$	System Under Learning
$\mathcal{T}$	Decision Tree, Regression Tree, or Hybrid Decision Tree
$\boldsymbol{A}$	System Matrix of an Ordinary Differential Equation
$\boldsymbol{B}$	Input Matrix of an Ordinary Differential Equation
B	Set of Basic Operators of Symbolic Regression (SR)
C	Set of Categorical Classes
E	Set of Edges of a Tree
F	Set of Flow Functions of a Hybrid Automaton or Hybrid Decision Tree
G	Set of Groups of Segments
I	Input Alphabet of a Mealy Machine
$\boldsymbol{J}$	Joined State and Input Matrix of an Ordinary Differential Equation

Symbol	Usage / Meaning
K	Set of Guards for the Transitions of a Hybrid Automaton
L	Language of an FSM
N	Neighborhood of a Symbol
O	Output Alphabet of a Mealy Machine
Q	Set of States of an Automaton
$\boldsymbol{S}$	Similarity Matrix for FaMoS
S	Set of Segments
T	Transitions of a Hybrid Automaton
V	Set of Nodes of a Tree
V_L	Set of Leaf Nodes of a Tree
X	State Space of a Hybrid Automaton or System
$\boldsymbol{Y}$	Observation Matrix of Autoregressive Models with Exogenous Inputs
c	Class Label of a Decision Tree
c_O	Penalty Factor in Segmentation Objective Function for the Identification of Hybrid Automata
d	Classification or Regression Function of a Decision Tree, Regression Tree, or Hybrid Decision Tree
e	Error between a Ground Truth and Predicted Value
f	Feature of Decision Trees, Regression Trees, or Hybrid Decision Trees
g	Performance Metric for Automata Forests
h	Inference Function for Automata Forests
i	Input Symbol
m	Number of Automata in a Forest
n	Length of Bounded History for Observations

Symbol	Usage / Meaning
o	Output Symbol
$\boldsymbol{p}$	Vector of Transition Points in the Identification of Hybrid Automata
p	Population Size of SR
q	Discretization Function
q_0	Initial State of an Automaton
r	Root Node of a Tree
$s_{\text{heuristic}}$	Configuration Parameter to Select the Heuristic in Test Case Generation with Decision Tree Models
$\boldsymbol{s}$	Trace of Samples
t	Time
t_S	Sampling Period
$\boldsymbol{u}$	Input Vector of a Matrix Ordinary Differential Equation
v	Node of a Tree
w	Map between Predicted and Ground Truth Points in the Identification of Hybrid Automata
$\boldsymbol{y}$	State Vector of a Matrix Ordinary Differential Equation
Σ	Alphabet of an Automaton
α	Output Function of a Mealy Machine
β	Transition Function of a Mealy Machine
ε	Error Bound for Decision Tree Models
$\bar{\zeta}$	Similarity Threshold of FaMoS
ζ_{dist}	Dynamic Time Warping (DTW) Distance
ζ_{diag}	DTW Alignment Correlation
ζ_{DTW}	Similarity Metric from DTW

Symbol	Usage / Meaning
ζ_{LMI}	Linear Matrix Inequality (LMI) Similarity Tolerance
η	Number of Generations of SR
θ	State Transition Function of a Deterministic Finite Automaton (DFA)
κ	Order of Differential / Difference Equation
λ	Maximum Evolution of a System within a Sampling Period
μ	Mean Value
ξ_G	Relaxation Coefficient of Grouping in the Identification of Hybrid Decision Trees
ρ	Ratio of Learning Data for Automata Forests
σ	Standard Deviation
φ	Similarity Threshold Factor of FaMoS
ψ	Parsimony Coefficient of SR
ω	Window Width
Ω	Objective Functions

Apart from the symbols listed in the table above, we use the following notation throughout this thesis:

- We use bold symbols to denote vectors, matrices, or traces.
- We use angle brackets $\langle \cdot, \cdot \rangle$ to denote tuples.
- We use square brackets $[\cdot, \cdot, \ldots]$ to denote vectors or lists and for indexing vectors, e.g., $\boldsymbol{y}[i]$.
- We use curly brackets $\{\cdot, \cdot, \ldots\}$ to denote sets.
- We use $|\cdot|$ to denote the cardinality of a set or the absolute value of a number.

Chapter 1

Introduction

Modern systems and processes are increasingly complex and interconnected due to the rapid advancement of digitization and the specialization of industry [1]. Models of these systems are essential for understanding the behavior and performance, as well as for ensuring dependability and safety [2]. For digital circuits, models and formal descriptions are used to verify the correctness of the design and to ensure that the system meets its specifications [3]. Models of manufacturing processes support the optimization of production or predictive maintenance, which reduces downtime and costs [4]. In the stock market, models are used to predict stock prices and optimize trading strategies [5]. These are just a few examples of the many applications where models play a pivotal role in understanding and optimizing complex systems.

The process of model generation is automated by learning models from data. This reduces the effort to create models for Cyber-Physical System (CPS)-driven applications [6]. This thesis focuses on model learning for technical systems. A CPS integrates physical components like sensors and actuators with digital logic [2] and, thus, covers typical systems in manufacturing and automation.

1.1 Cyber-Physical Systems in the Context of Digitization

E. A. Lee et al. [7] introduce CPSs as systems that integrate computation with physical processes. CPSs are at the intersection of the physical and digital world. An inherent characteristic is the interaction and integration of discrete layers such as control logic and continuous process layers, e.g., representing sensor or system variables [1]. This

complexity is compounded by the fact that CPSs are often subject to external influences, such as environmental conditions or user interactions, which can lead to non-deterministic observations. An illustrative example incorporating these characteristics is a heating system. While the discrete process is the decision to turn the heating system on or off, the continuous process is the temperature of the room. The temperature is influenced by external factors such as the outside temperature, the number of people in the room, or other heat sources like electrical devices. This interaction between discrete and continuous processes is a key characteristic of CPSs and poses challenges for modeling and understanding the system behavior which are addressed in this work. A further characteristic of CPS is the composition of several components. While this characteristic creates challenges in the context of networked systems [8], we consider the system as a whole and rather abstract from the individual components.

We create abstract models of CPSs, which are needed to understand the behavior of the system. For example, a purely discrete system model abstracts from the continuous process and only considers the discrete process [9], [10]. The discrete model has a finite state space, which allows a comprehensive verification of the correctness of the system with respect to behavioral requirements [11]. For monitoring and control, a more powerful abstraction is needed. The model must include the continuous process and, thus, is a hybrid model [12] or purely continuous model [13]. The range of model types and abstractions is vast and the choice of model type depends on the application and the requirements of the system [14], [15], [16].

With this work, we focus on the abstraction and modeling of CPSs from a data-driven perspective. With a data-driven approach, we aim to automate the modeling process and eliminate the effort required to create accurate and adequate models manually. The traditional, manual identification [17], [18] of the physical behavior of CPSs is often infeasible and prone to errors from unconsidered influences. Data-driven modeling finds a model from observations of the system and considers the actual observable behavior without biased interpretation. Instead of machine learning methods such as deep learning, this work focuses on the learning of explicit models that are interpretable and match with the intuitive understanding of the system [19]. While interpretability of models is an independent field of research, we consider models interpretable if they can be understood by a human and provide insights into the system behavior. In this sense, discrete models are particularly manageable and interpretable [11], [20], [21].

The range of learning methods as well as the range of model types and abstractions is vast and ranges from abstract logical models, e.g., temporal logic [22] over discrete models like deterministic finite automata [10] and decision trees [23] to hybrid models like hybrid automata [12] and purely continuous and non-linear regression models [13]. In this work, we focus on three types of models and their learning methods:

1) automata learning techniques, which we specifically analyze for CPS,
2) a novel application of Decision Tree Learning (DTL), which we extend with formal learnability guarantees that establish theoretical foundations, and

3) pioneering algorithms for hybrid automata learning that enable an integrated modeling of discrete and continuous dynamics.

A learned model serves as the foundation for many applications such as model-based testing [9], behavioral prediction [13], diagnosis [16], monitoring [24], or explanation [21]. For monitoring, a model enables validation of system conformance, where high accuracy and robustness to unseen data are crucial to ensure confidence in the results. In testing, models support the generation of test cases that aid both system design and maintenance. Prediction uses the model to forecast system behavior in various environments, which is particularly valuable for CPSs where in-field testing may be costly or constrained. For diagnosis, the model helps to identify discrepancies between expected and actual behavior, potentially providing insights into faults. Finally, explanation relies on interpretable models to make system behavior understandable for users or other systems. Especially, if the system initially lacks transparency or is perceived as a black box. We present a range of applications for our proposed learning methods, which showcase the versatility and applicability of data-driven modeling in various contexts.

1.2 Motivating Example for Modeling Challenges in CPS

To illustrate the challenges and opportunities in data-driven modeling of CPSs, we present a motivating example alongside which we introduce the research questions addressed in this thesis and reveals limitations of model learning for CPS. This example from S. Plambeck et al. [25], based on a real-world Light Detection and Ranging (LiDAR) localization system, demonstrates how data-driven modeling supports the analysis and integration of CPS and highlights the gaps that our proposed methods aim to fill.

1.2.1 Challenge 1: Identification of Discrete System Models

We consider a logistics scenario where a LiDAR Real-Time Localization System (RTLS) is installed on a mobile robot, e.g., on a forklift for autonomous navigation in a warehouse as shown in Figure 1. This system exemplifies typical CPS characteristics: it integrates physical sensors such as lasers and light detectors with digital processing algorithms, e.g., Simultaneous Localization and Mapping (SLAM) algorithms [26]. Further, the LiDAR system operates in dynamic environments with varying conditions such as logistics warehouses with changing layouts and lighting conditions, and must provide reliable performance for safety-critical applications.

The performance of the system depends on multiple external influences and configuration parameters, where we consider the following parameters for this example:

Figure 1 — Illustration of the motivating example: a LiDAR localization system on a forklift in a logistics warehouse.[1]

- Reflector support: reflecting surfaces mark positions in the environment. The value of this parameter is either *on*, if the reflecting surfaces are present, or *off*, if they are not present.
- Inertial Measurement Unit (IMU) support: an inertial measurement unit provides motion information. The value of this parameter is either *on*, if the IMU is used, or *off*, if it is not used.
- Map quality: accuracy of the pre-recorded environment map, which we classify as good, medium, or bad, depending on the degree of deviation between the actual environment and the map,
- Field of view: angular extent of the LiDAR perception, which is a continuous value between 0° and 360°.

Different combinations of parameter valuations lead to various configurations that impact system performance.

The robot should navigate safely for different application scenarios to pick up and transport goods of varying sizes. From these scenarios, we derive different requirements defined as localization accuracy categories that are application-dependent, categorized as *small object-precise* ($< 20\,\text{mm}$), *medium object-precise* ($20\,\text{mm} - 50\,\text{mm}$), and *storage box-precise* ($50\,\text{mm} - 200\,\text{mm}$).

A learned model for this robot supports the identification of relevant influences on the performance of the localization, describes the dependency between the configuration parameters and the localization accuracy, and provides explanations and insights into the system behavior.

Such a learned model represents the static mapping from the external influences to the categories of localization accuracy. A challenge remains the representation of

[1] Image generated with ChatGPT

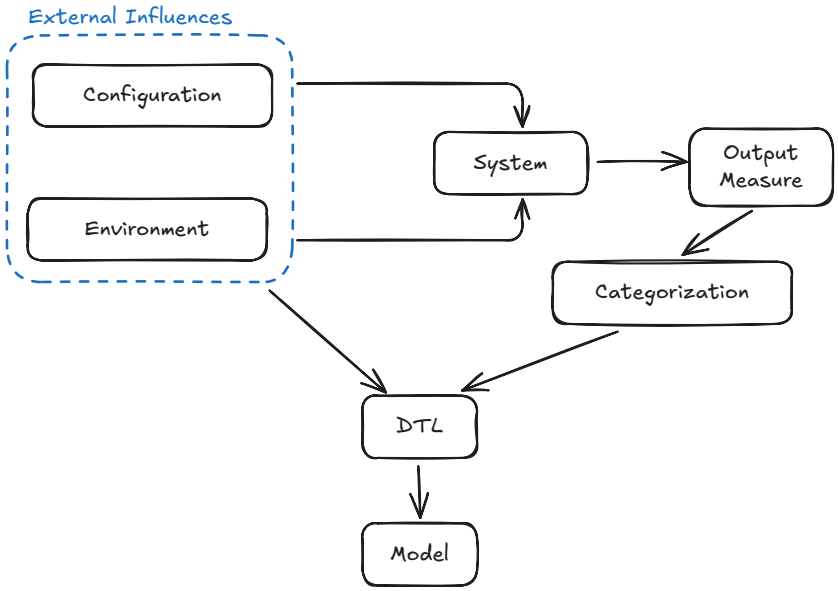


Figure 2 — Modeling process for the RTLS: transforming continuous measurements into categorical predictions. DTL finds a model, which describes the dependency between the external influences and the localization accuracy.

system processes that capture behavior over time, e.g., modeled using Finite State Machine (FSM). This observation leads to our first research question:

> **Research Question 1**: In how far are classical learning methods for FSMs suitable for modeling CPSs?

This question is addressed in Chapter 4, where we apply classical automata learning for modeling CPSs.

1.2.2 Challenge 2: Integration of Temporal Information

Omitting technical details, here, we explore DTL as a modeling approach that classifies the category of localization accuracy based on the observed configurations and environmental conditions.

Figure 2 shows the conceptual procedure for data-driven modeling of the dependencies between the external influences and the output of the system. The external influences form the features for DTL. Having the influences as an input, we observe the system and collect the output, i.e., the localization error. The measurement of the localization error is a continuous value representing the difference between the estimated and the true position of the robot. As the decision tree should predict the localization quality in terms of the categories given by the application-based accuracy categories, *small object-precise*, *medium object-precise*, and *storage box-precise*, a mapping from the

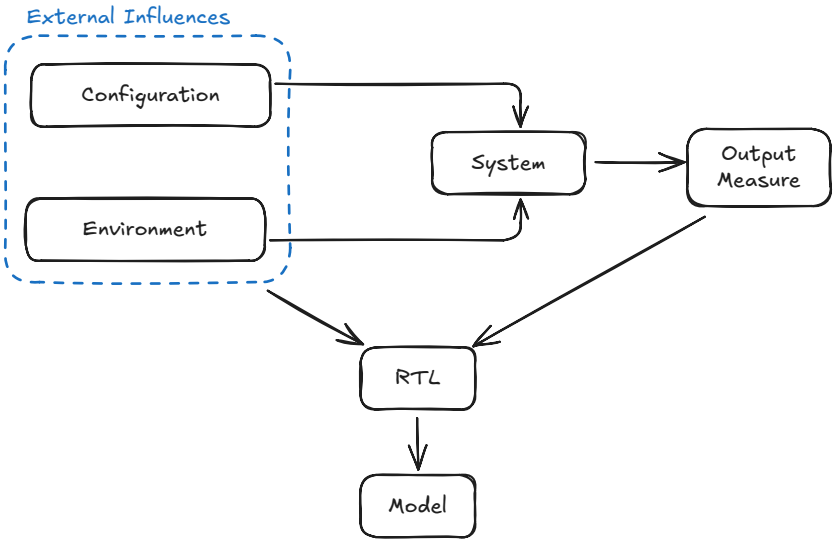


Figure 3 — Modeling process for the RTLS with RTL: directly predicting continuous localization errors without categorical abstraction.

continuous error values to the accuracy categories is required. This categorization is an abstraction from the continuous output space to a set of categories.

The categorized outputs together with the information on external influences are input to DTL. Nevertheless, for a stateful system, past information is relevant and should be integrated as features for DTL. This motivates our second research question:

Research Question 2: What is the performance of decision trees that integrate temporal information for modeling of CPS?

This question is addressed in Chapter 5, where we apply DTL using a temporal window to integrate past observations as features.

1.2.3 Challenge 3: Capturing Continuous & Hybrid Dynamics

As an alternative to categorization, we apply an approach that preserves the continuous nature of the localization measurements. We explore Regression Tree Learning (RTL) to learn a regression tree for the continuous output of the LiDAR localization system. The regression tree predicts continuous output values directly, eliminating the need for categorical abstraction.

Figure 3 shows this alternative modeling procedure. In contrast to Figure 2, the categorization step is omitted as no categorization is needed for RTL. While RTL accounts for the continuous nature of the output measure, this approach reveals a further challenge. Many CPSs actually exhibit hybrid dynamics, where both continuous and

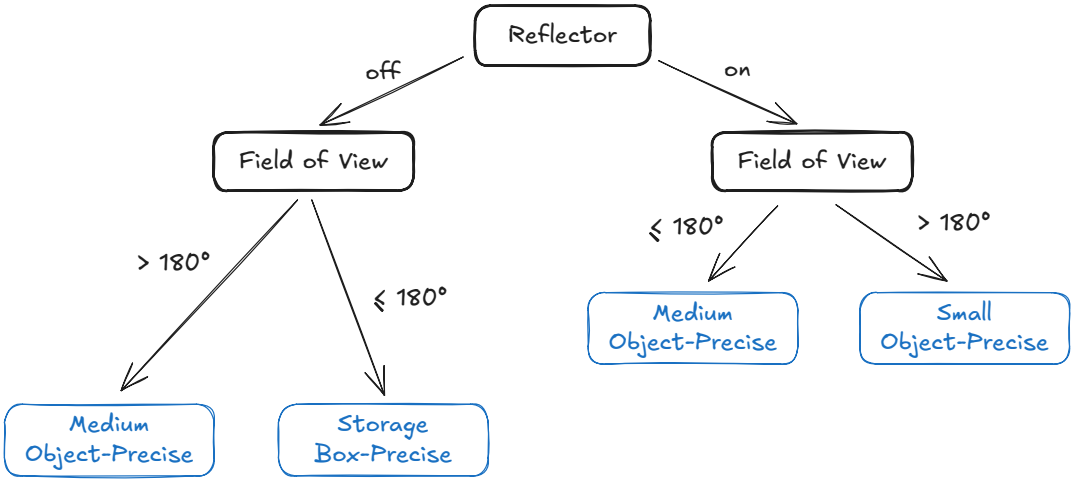


Figure 4 — Decision tree model for application-related categories (storage box-precise, medium object-precise, and small object-precise).

discrete changes occur. This limitation motivates our exploration of more sophisticated modeling approaches. Thus, we have the third research question:

> **Research Question 3**: How to learn a model to capture hybrid dynamics of CPSs?

This question is addressed in Chapter 6, where we present algorithms for the identification of hybrid automata that enable an integrated modeling of discrete and continuous dynamics.

1.2.4 Challenge 4: Evaluation & Comparison of Modeling Approaches

In the final step of our motivating example, we present empirical results from applying DTL and RTL to the LiDAR localization system.

Figure 4 shows a decision tree model learned from the LiDAR system data. The model reveals that reflector support and field of view are the most critical factors for achieving precise localization, while other features play secondary roles. This demonstrates how a learned model automatically identifies relevant system influences and provides explanations of system behavior. Models, like decision trees, that offer such insights are referred to as interpretable models, which are a central focus of this thesis.

However, when applying regression tree learning to predict continuous localization errors directly, we observe different behavior as shown in Figure 5. The regression tree model tends to overfit on the limited training data. This is, e.g., visible from the two leaves with very similar mean error of 15 mm and 16 mm, respectively, which are probably not distinguishable in practical, noisy scenarios.

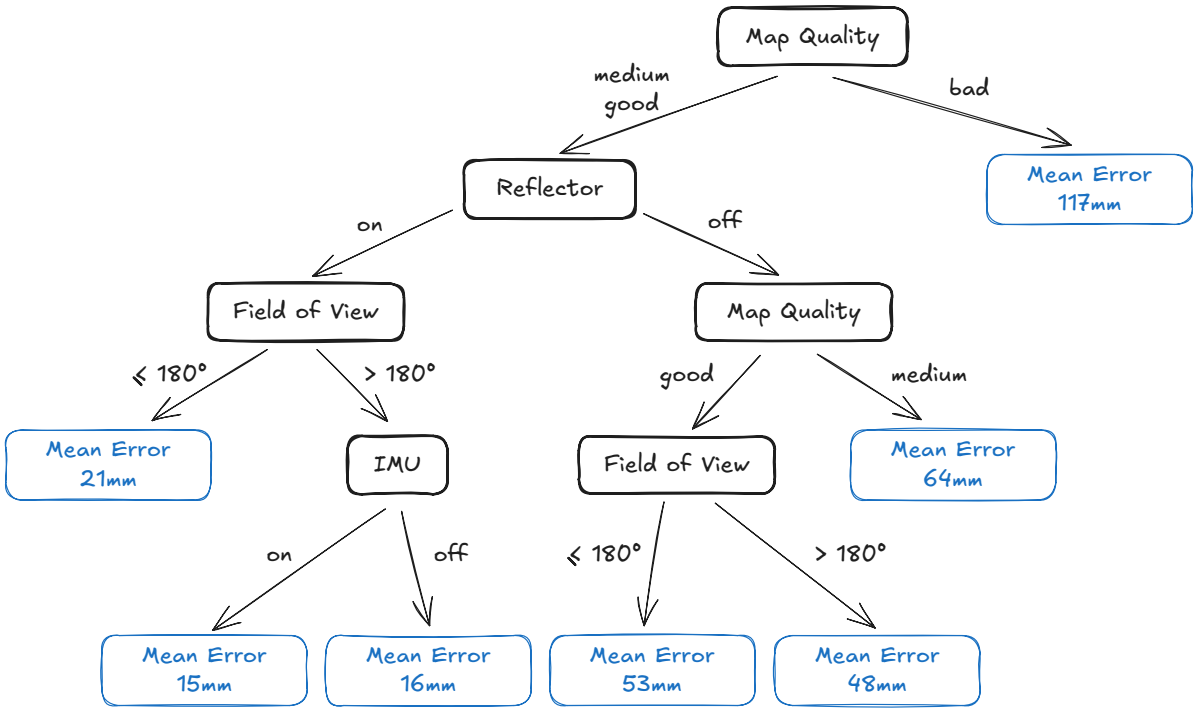


Figure 5 — Decision tree model with RTL. The predicted localization error is shown in the leaves of the tree.

This comparison illustrates that different modeling approaches reveal different aspects of system behavior and exhibit distinct trade-offs between interpretability, generalization, and expressiveness. At the same time, this analysis demonstrates how learned models provide explanatory insights for specific application scenarios. The models reveal critical factors influencing localization quality in the LiDAR system and find the relative importance of different features for achieving precise localization.

Comparing the two approaches reveals distinct trade-offs between interpretability and accuracy. While the decision tree model offers structured, rule-based explanations of system behavior, the regression tree provides more detailed feature importance rankings but suffers from overfitting on limited data. This comparison illustrates that model selection extends beyond performance metrics to encompass qualitative aspects such as interpretability and robustness. These findings motivate our fourth research question:

Research Question 4: How do different model types and learning strategies for CPSs compare in terms of their characteristics, e.g., interpretability and accuracy?

This question is addressed in Chapter 7, where we compare the modeling strategies of this work with respect to their characteristics and applicability to CPSs as well as their modeling accuracy.

1.2.5 Insights and Research Gaps

This comprehensive analysis of the LiDAR localization system reveals both the potential and fundamental limitations of current modeling approaches for CPSs. The results demonstrate that data-driven modeling successfully

- identifies relevant influences on system performance automatically (feature importance analysis),
- reveals correlations between environmental influences, external parameterization, and performance, and
- generates interpretable explanations for system behavior through decision tree structures.

However, the empirical results also expose critical limitations that motivate our research contributions, which are

- the representation of processes and CPS dynamics over time,
- the integration of temporal information into model training and inference,
- the development of new model types and learning algorithms to integrate discrete and continuous dynamics, and
- a comprehensive comparison of different modeling strategies for CPSs considering quantitative and qualitative aspects.

These insights motivate our research to address the identified limitations and improve the modeling of CPSs.

1.3 Contributions & Structure

With the illustrative example in the previous section, we identify four open research questions in the field of data-driven modeling and learning of CPSs, which we address in this work:

Q1 In how far are classical learning methods for FSMs suitable for modeling CPSs?

Q2 What is the performance of decision trees that integrate temporal information for modeling of CPSs?

Q3 How to learn a model to capture hybrid dynamics of CPSs?

Q4 How do different model types and learning strategies for CPSs compare in terms of their characteristics, e.g., interpretability and accuracy?

In response to these research questions, this thesis proposes an exploration of data-driven modeling strategies for CPSs, progressing from purely discrete to hybrid representations with increasing expressiveness. The common foundation of all modeling strategies is learning abstract models from system observations, as shown in Figure 6. However, each approach addresses different aspects of the fundamental challenge in CPS modeling: balancing interpretability with the ability to capture the complex discrete-continuous interactions inherent in these systems.

This work follows a deliberate progression motivated by the limitations revealed in the LiDAR case study. Starting with completely discrete and deterministic models,

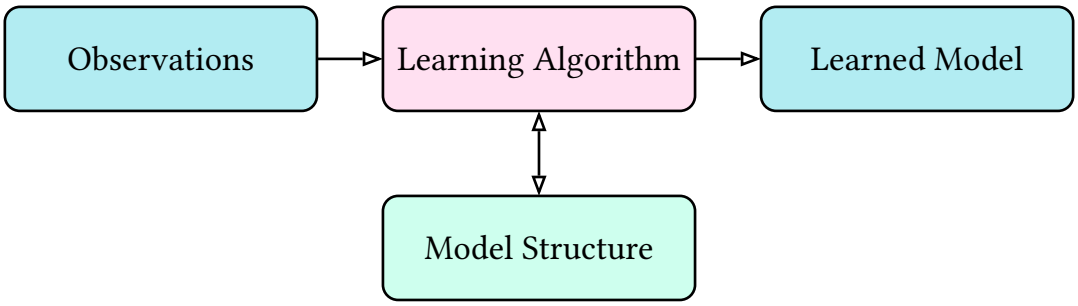


Figure 6 — Data-driven model learning: the observations of the system are used to learn a model. A learning algorithm is applied to the observations. The learning algorithm and the model structure have a mutual dependency — either a learning algorithm for a specific model structure or a generic learning algorithm that applies to different model structures is used.

we systematically explore how to handle increasing complexity while maintaining practical applicability. The entry point to modeling, as shown in Figure 6, are always the observations of the system, which the learning algorithms process for each model structure specifically. Each learning algorithm produces a model that represents a different abstraction of the system behavior, trading off interpretability, expressiveness, and computational complexity. The model structure describes the type of model and formalism used, e.g., a Deterministic Finite Automaton or a decision tree. The learned model is a concrete instance of the used formalism representing the behavior of a specific system at a certain level of abstraction. Related work on the general topic of model learning is presented in Chapter 2, while subsequent chapters contain individual sections on related work in their respective fields.

We provide and explore three complementary modeling strategies that represent a systematic progression through the modeling space for CPSs:

1) **Finite State Machines (FSMs)**: Automata learning techniques specifically tailored for CPS environments, providing interpretability and formal verification capabilities,
2) **Decision Trees**: Novel DTL algorithms with formal learnability analysis that establish theoretical foundations while handling non-deterministic behavior, and
3) **Hybrid Automata**: Pioneering algorithms for hybrid automata learning that bridge the discrete-continuous modeling gap inherent in CPSs.

This sequence of models reflects the evolution of modeling needs identified in our LiDAR case study: starting from purely discrete abstractions, moving to models that capture non-deterministic behavior, and finally to hybrid models that can capture the full complexity of continuous-discrete interactions. Each approach builds upon insights from the previous one while addressing its fundamental limitations.

Chapter 3 introduces preliminary concepts for the three modeling strategies and introduces examples that are used throughout the work. Figure 7 shows the learning methods and model structures, which aligns with the structure of this work.

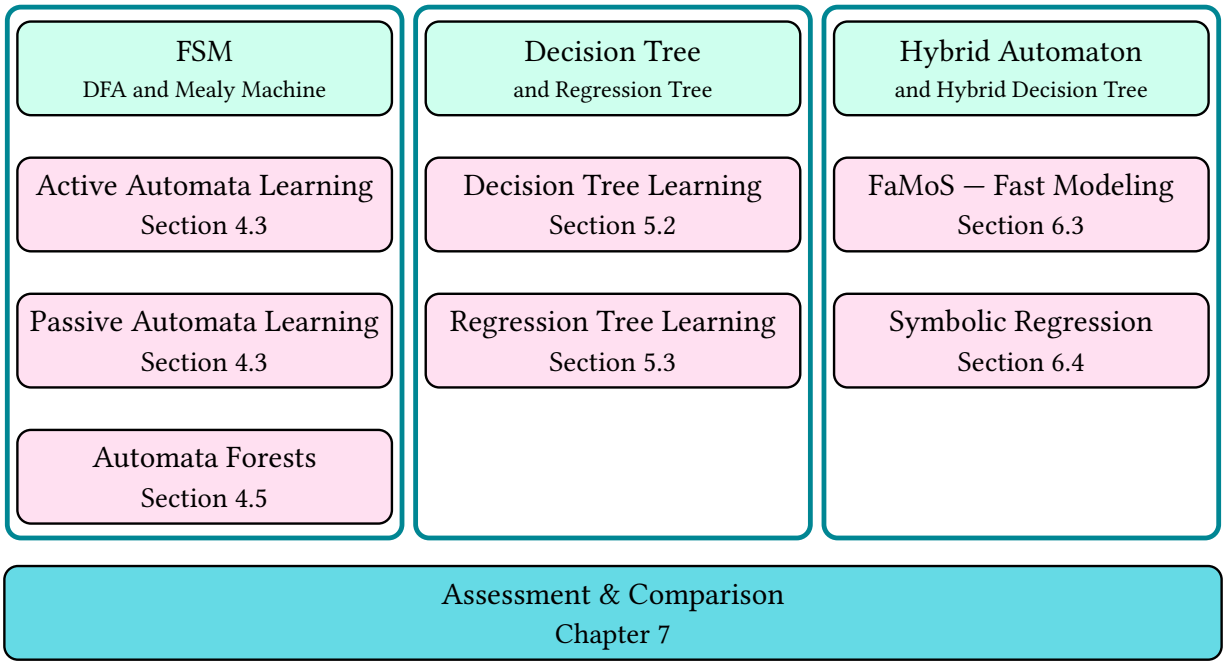


Figure 7 — Three types of model structures (green) FSMs, decision trees, and hybrid automata are considered in this work. The respective learning algorithms (pink) for these model structures are shown per column. A joined comparison and discussion accumulates the results.

Table I — Overview of the research questions and their corresponding chapters and publications.

Research Question	chapter	Publications
Q1	Chapter 4	[27], [28], [29]
Q2	Chapter 5	[30], [31], [32], [33]
Q3	Chapter 6	[34], [35], [36]
Q4	Chapter 7	[37]

Every modeling strategy is covered in a separate chapter and addresses one of the research questions. Table I shows the related chapters for the research questions. Further, the publications related to this thesis are listed in the table according to the research questions.

Chapter 4 addresses the first research question Q1. We present a modeling strategy that learns FSMs, specifically DFAs from a System Under Learning (SUL), i.e., the system that is being observed and learned from. This approach provides the founda-

tion of our modeling hierarchy by establishing how classical automata learning can be adapted for CPSs. We integrate published results from L. Schammer et al. [27] and S. Plambeck et al. [28], which show that classical automata learning techniques are applicable to CPSs. We extend this by a discussion on suitable abstractions of the system. Although techniques to achieve such an abstraction are not the main focus of this work, we provide an idea on the meaning of abstraction in the context of model learning and some straightforward techniques to achieve such an abstraction. These DFA models are purely discrete and deterministic. These characteristics make them interpretable and easy to verify. We focus on discrete and deterministic methods for learning models of CPSs in the form of DFAs or Mealy machines. Classical automata learning algorithms are analyzed for their applicability to CPSs and applied for test case generation and system monitoring. Although automata learning has a strong fundamental theoretical base including learnability results, the learned model highly depends on the quality of the input data. Thus, we propose an extension of the classical learning algorithm to improve the robustness of the learning process under uncertainty in the input data, which is published by A. Krumnow et al. [29].

A suitable deterministic abstraction of CPSs is not always possible or known, as demonstrated in our LiDAR case study, where an application-dependent categorization is not necessarily available. Chapter 5 addresses the second research question Q2 by evaluating the applicability of decision trees as an alternative discrete model for CPSs that can handle non-deterministic behavior while maintaining interpretability. Decision trees are a well-known and widely used model for discrete systems [38]. We propose DTL as a method for learning models of CPSs by S. Plambeck et al. [30] and further analyze their capabilities by S. Plambeck et al. [31] and S. Plambeck et al. [32]. We integrate the previous results into a joined formalism, which generalizes our formal results on DTL. Decision tree models predict a discrete output based on past observations of the system and are not restricted to deterministic behavior. A major contribution is to draw the connection between decision trees and DFAs, showing how decision trees are viewed as an extension of finite automata that handle non-deterministic behavior. We realign decision tree models with DFAs and identify a fundamental limitation of model learning under finite observations. Further, we propose a novel strategy for test case generation based on decision tree models as presented by S. Plambeck et al. [33].

Finally, we propose a third modeling strategy that learns hybrid automata from a System Under Learning (SUL) in Chapter 6 — answering the third research question Q3. While decision trees can handle discrete non-determinism, they still abstract away the continuous dynamics that are fundamental to CPSs, as observed in the continuous localization errors in our case study. In contrast to the first two modeling strategies, hybrid automata incorporate continuous dynamics and, thus, are able to model both, discrete and continuous behavior of a system within a single, unified framework. Two new data-driven learning algorithms are proposed for learning hybrid automata from data, which are published by S. Plambeck et al. [34] [36]. The first algorithm focuses

on an efficient and fast model identification with intuitive identification methods. The second algorithm challenges the traditional learning process for hybrid automata by identifying the discrete structure of the system based on the underlying continuous dynamics.

The fourth research question Q4 is addressed in Chapter 7. This final analysis integrates our progression from discrete-deterministic through discrete-probabilistic to hybrid models. We present a framework for comparison and integration of general modeling strategies for CPSs. Considering this, we compare the three modeling strategies of this work with respect to their characteristics and applicability to CPSs as well as their modeling accuracy. This goes beyond the related publications and provides a comprehensive overview of the modeling strategies, identifying when each approach is most suitable and how they complement each other in practical applications. The comparison reveals that the choice of modeling strategy depends on the specific characteristics of the CPS, the available data quality, and the intended application requirements. Every model strategy is complementary and shows superior performance in different scenarios. Automata learning leverages exact modeling, identifying the original ground truth structure of the system on sufficient learning data. Automata forests and decision tree models improve robustness on limited datasets and provide better generalization capabilities to practical learning scenarios. Hybrid automata extend this by incorporating both discrete and continuous dynamics, allowing for a more expressive and comprehensive representation of complex systems.

A summary of the work and an outlook on future work is given in Chapter 8. This thesis, thus, provides a comprehensive exploration of data-driven modeling for CPSs, establishing both theoretical foundations and practical methodologies that advance the field from purely discrete abstractions to integrated hybrid representations, all while maintaining interpretability and expressiveness.

Chapter 2

Related Work

There are many fields such as control theory, computer science, and discrete mathematics that deal with the modeling of systems. The formal description and theoretical foundation of these models and systems are different in each field [39]. Still, the need for models leads to several approaches that create a more abstract model or specification of a system and, thus, relate to this work. Many of these works consider CPSs as a synonym for complex systems involving digital computing units. A general overview of applicable methods and underlying challenges is given by P. Derler et al. [40], which identifies major challenges in non-deterministic behavior, consistency of models, complexity and size of models and systems, distributed behavior, and system heterogeneity.

This chapter covers two perspectives: we pinpoint exemplary works from various research directions of the field addressing the fundamental challenge of balancing model accuracy with interpretability. We analyze approaches ranging from comprehensive system representations to abstract specifications, identifying their strengths and limitations in the context of interpretable model learning. Further, we review existing modeling approaches for CPSs under the aspect of learnability through a comprehensive structured literature review. Every preceding chapter contributes further, deeper related work for their specific topics and focus.

2.1 Related Fields in Model Identification

Several fields contribute to the identification and modeling of systems, each with its own methodologies and focus areas. These fields include:

- Digital Twins: focus on creating virtual replicas of physical systems to enable, e.g., real-time monitoring and simulation,

- Abstraction: emphasizes the creation of simplified models that capture essential system behaviors while ignoring irrelevant details,
- Manual Modeling: involves the traditional approach of manually constructing models based on expert knowledge and system understanding,
- Machine Learning: leverages data-driven techniques to automatically learn models from data, often improving adaptability and performance,
- Physics-Inspired Learning: integrates physical laws and principles into the learning process to enhance model accuracy and interpretability,
- Finite State Machines & Hybrid Automata: utilize formal methods to model systems with discrete and continuous dynamics, and
- Probabilistic Modeling: incorporates uncertainty and variability into the modeling process, allowing for more robust and adaptable models.

We pinpoint exemplary works from these fields and identify open challenges, which we address in this work.

2.1.1 Digital Twins

Models of complex systems are also widely discussed under the term *digital twin*. Digital twins often represent an accurate model of a system that can, e.g., be used for holistic simulations [41]. The holistic model often consists of multiple sub-models. Digital twins have a broad range of applications which lead to various parallel definitions and interpretations of the term [42]. We showcase exemplary works that illustrate the diversity of digital twin implementations with a focus on CPS modeling.

An example of a digital twin framework is described by M. Müller et al. [43], which improve safety of an autonomous truck with an adaptive digital twin. J. Jeon et al. [44] propose a framework designed to develop trustworthy digital twins of a CPS with a particular focus on marine engines. The authors claim that model learning for systems often suffers from a lack of learning data, which hinders the deployment of models for fault identification and anomaly detection. The proposed digital twin is employed for data generation, aiding model learning for anomaly detection, identification, and isolation efforts. The data generation procedure defines a set of scenarios that the digital twin executes to produce a learning dataset. Subsequently, a model for fault detection, identification, or isolation is trained on the generated data and evaluated to ensure its performance.

In the work of F. Vitale et al. [45], model refinement is employed within the context of generating a digital twin for an industrial CPS using Petri nets. The proposed methodology features both offline and online learning phases. During the offline phase, the Petri net model is iteratively refined based on historical data. Furthermore, the offline phase applies a conformance check, which decides whether the iterative learning is continued.

While digital twin approaches provide comprehensive system representations, they typically focus on high-fidelity simulation rather than interpretable model structures.

Thus, there is a need for learning approaches that balance model accuracy with interpretability, enabling both predictive capabilities and explainable insights into system behavior. This work addresses this gap by developing interpretable modeling strategies that provide transparent decision-making processes while maintaining sufficient accuracy for practical applications.

2.1.2 Abstraction Methods & Abstract Models

While the field of digital twins emphasizes high-fidelity representations, this work focuses on developing models on a compact, abstracted level that are interpretable and generate explainable results. We define an abstraction of a system a priori, but there exist also approaches for an automated abstraction of a system. Approaches for alphabet abstraction refinement are shown by F. Howar et al. [10] and F. Aarts et al. [46], where the learner is in control of the abstraction of the system. Abstraction refinement has been applied to timed automata, a sub-class of hybrid automata by V. Roussanaly et al. [47]. In their scenario, abstract zones over time are defined that result in the states of the learned automaton.

Several modeling approaches directly utilize abstract specifications of the system, such as reachable sets or temporal logic specifications. For digital circuits, models and formal descriptions are used to verify the correctness of the design and to ensure that the system meets its specifications [3]. A. Devonport et al. [48] introduce a strategy for data-driven reachability analysis. The learned reachable set can be considered as a model to characterize and verify the behavior of safety-critical CPSs. The approach uses a Probably Approximately Correct (PAC)-bound, which provides probabilistic guarantees of model accuracy. PAC learning [49] originally discusses classes of learnable problems based on Boolean expressions. A program is considered learnable if there exists a polynomial-time algorithm that learns an approximation of the program that guarantees an error bound with high confidence, i.e., high probability. For Boolean expressions, this means the learned approximation has no false positives (outputs true when it should be false) while maintaining high true positive rates (outputs true in most cases where it should). This PAC formulation finds a reachable set, which probabilistically matches a ground truth reachable set with a given confidence, acting as both a learning mechanism and an evaluation metric. The authors propose two approaches for determining this PAC bound: convex scenario optimization and empirical risk minimization.

In addition to that, G. Saveri et al. [22] provide an approach to gather Signal Temporal Logic (STL) formulae from time-series data by integrating Bayesian optimization with information retrieval techniques. The method is designed to identify STL formulae that effectively distinguish between nominal and anomalous traces. The authors propose constructing a continuous latent space that captures the similarity of STL formulae. The authors develop an algorithm capable of mining STL formulae from labeled data, where the labels denote whether the system adheres to a desired behavior.

While abstract specifications like reachable sets and temporal logic provide formal guarantees, they often lack direct applicability to practical CPS. This work addresses this limitation by developing learning approaches that produce directly applicable models for practical CPS applications while maintaining interpretability.

2.1.3 Manual Modeling

Traditional modeling approaches design models manually. One conventional approach to model a CPS is to describe the physical relationships as a classical system of formulas [17], [18]. By performing numerical simulations, the behavior of a CPS can be determined over time.

Simulation and modeling can, e.g., be done with Matlab and Simulink [50] or Modelica [51]. Another established language for modeling is SysML [52] that is a Unified Modeling Language (UML) profile, where UML is commonly used in software, and allows defining several properties like requirements, response time, and functionalities. A SysML representation might be the basis to design systems or their simulations and support test case generation and verification. There also exists an extension for SysML to model continuous physical interaction and signal flows [53].

Manual modeling requires that the internals of the system are sufficiently and accurately known. However, this is often not the case for CPSs, where the system under learning consists of multiple sub-components whose internals are not fully known, and the behavior depends on physical, environmental influences, where the identification of relevant factors is challenging.

Manual modeling approaches face fundamental limitations when dealing with complex CPSs that exhibit non-deterministic behavior or when complete system knowledge is unavailable. Thus, there is a need for automated learning approaches that extract meaningful models from observational data without requiring detailed prior knowledge of system internals. This work tackles this challenge by developing data-driven learning strategies that automatically infer interpretable models from system observations, accommodating both deterministic and non-deterministic behaviors.

2.1.4 Machine Learning

Data-driven approaches are useful alternatives, especially for complex systems, and replace time-consuming manual modeling. Similar to digital twins, machine learning is a wide term and covers various techniques for modeling and simulation not only for physical systems, but also for signal and image processing [54], natural language processing [55], analysis of biological data [56], and more. More generalistic machine learning approaches combine multiple models through ensemble learning techniques [57]. Similarly, approaches from AutoML [58] aim to automatically select and optimize machine learning models for a given task.

Classical machine learning methods include decision trees, support vector machines, and neural networks and focuses on the identification of patterns in data. Methods from the field of machine learning such as neural networks are successfully used for modeling CPSs, such as production lines, robots, and embedded systems [59], [60]. H. Steude et al. [61] apply representation learning and deep learning for CPSs. Methods based on reinforcement learning or similar active learning paradigms interact with a system to improve the exploration of the behavior and the stability of the system [62]. However, deep learning approaches usually provide only limited interpretability and explanation of the (physical) system behavior [19].

Many machine learning approaches for CPSs such as neural networks typically prioritize predictive accuracy over interpretability, resulting in black-box models that provide limited insights into system behavior. This creates a fundamental challenge for applications requiring explainable decisions, such as safety-critical monitoring and fault diagnosis. This work directly addresses this interpretability gap by developing learning approaches that inherently produce transparent, interpretable models while maintaining competitive accuracy for CPS applications. We utilize classical machine learning techniques which are interpretable, such as decision trees [20], but also explore more advanced methods. Instead of the identification of patterns, we focus on the extraction of structured knowledge from data.

2.1.5 Physics-Informed Learning

Physics-inspired learning is a paradigm that integrates physical knowledge into the learning process. Z. Wang et al. [63] present an example of a physics-informed or physics-inspired learning approach. This can be done by using physical models to guide the learning process, or by incorporating physical constraints into the learning algorithm.

E. Kiyani et al. [64] present a framework for physics-informed learning to determine a model of equations of motion. The method combines symbolic regression with physics-informed neural networks. The authors call this a gray-box approach, because partial knowledge of the equations of motion is used to guide the learning process.

An overview of physics-informed learning is given by G. E. Karniadakis et al. [65]. The authors focus on physics-informed neural networks and their applications. A major advantage of physics-informed learning exists in case of noisy, limited, or incomplete learning data.

Z. Wang et al. [63] integrate physics-based information into neural network training, which enhances the fidelity of models by grounding data-driven processes in knowledge about the physical world. The presented method uses the physical knowledge to derive the governing equations of the system. Then, a neural network is trained that learns the remaining, unknown relationships and parameters. The key aspect in the adaptation of the learning process is that the loss function does not only minimize

the reconstruction on the training data, but also minimizes a physical loss, which evaluates the known physical equations.

While physics-informed learning approaches successfully integrate domain knowledge into the learning process, they often require extensive prior knowledge of the underlying physical principles and may not generalize well to systems where such knowledge is incomplete or unavailable. This work complements physics-informed approaches by developing learning strategies that extract interpretable models from observational data even when detailed physical knowledge is limited, providing alternative modeling pathways for complex CPSs.

2.1.6 Finite State Machines & Hybrid Automata

In this work, we focus on the learning of discrete models, such as DFAs, Mealy machines, and hybrid automata. Approaches that learn formal languages of DFAs facilitate a deep theoretical understanding of their limitations and their capabilities. Limitations of learning languages from examples have been well-studied, starting with the seminal work of E. M. Gold [66]. The learning approaches split into active [67], [68], [69] and passive [70], [71] automata learning.

Moreover, switching dynamical systems form a subset of hybrid automata which focus on the detection of switches in continuous dynamics, while the modeling of complex mode transitions is less considered [72]. Hybrid automata [12] are an extension of DFAs that combine discrete and continuous dynamics. Model learning strategies for hybrid systems usually split the learning problem into sub-problems for the discrete and continuous parts. The range of methods for both sub-problems is large and includes, e.g., wavelet analysis and support vector machines [73] or clustering methods [24]. A restricted version of hybrid automata considering only time as an additional variable is called timed automata. O. Maler et al. [74] present an approach to approximate continuous systems by timed automata is presented. Further, M. Tappler et al. [75] use genetic programming to passively learn a timed automaton model of a system.

Existing automata learning approaches typically focus on either purely discrete systems or require significant assumptions about system structure and data characteristics. In this work, we extend from classical automata learning to decision trees and hybrid automata, enabling systematic comparison and appropriate model selection based on system characteristics and application requirements.

2.1.7 Probabilistic Models

A further extension of model learning for CPSs is the incorporation of probabilistic models. A relevant paradigm for probabilistic modeling is PAC learning [49] as introduced above. PAC learning strategies provide a confidence bound on the learning process. While originally applied to Boolean expressions, M. J. Kearns [76] extend this

concept to weak and strong PAC learning. Furthermore, learning in the presence of noise is considered, which describes learning Boolean expressions under presence of noise, i.e., with inexact oracles and statistical queries. A similar analysis is presented by D. Angluin et al. [77], which determines, e.g., the minimum amount of undisturbed samples required for successful learning.

Another approach in probabilistic modeling are stochastic models such as probabilistic or non-deterministic finite automata and Markov models. A first attempt to learning stochastic languages by finding an automaton representation is given by R. Carrasco et al. [78]. In Chapter 16 by C. de la Higuera [79], algorithms for learning probabilistic, deterministic finite automata are discussed. D.-Y. Yeung et al. [80] use the Baum-Welch algorithm to learn hidden Markov models based on fixed-length traces of events for intrusion detection on shell command or system call data. H. Mao et al. [81] present a learning algorithm for probabilistic automata in form of labeled Markov chains based on data from a black-box system.

Probabilistic models provide valuable uncertainty quantification, but they often increase model complexity and may reduce interpretability compared to deterministic approaches. This work contributes to this area by developing deterministic interpretable models that handle non-deterministic system observations through over-approximation strategies, providing a complementary approach to probabilistic modeling that maintains transparency while accommodating uncertainty in system behavior.

2.2 Learnability - A Systematic Literature Review

While this work primarily focuses on a progression of methods for the identification of interpretable models of CPSs, another aspect is the integration of domain and expert knowledge into the learning process. M. Schmidt et al. [82] present a literature review on the learnability of models for cyber-physical systems, highlighting the importance of incorporating prior knowledge and constraints into the learning process to improve model performance and interpretability. By addressing the learning procedure in a structured way, our literature review is a relevant complementary analysis to the presented related work, providing insights into a broad range of learning methods and their applicability to cyber-physical systems.

2.2.1 Methodology

Our methodology for the literature review is based on a systematic approach to identify, analyze, and synthesize relevant works in the field of learnability for CPS. The basis of our review is the research question *Which formal statements, guarantees, results and approaches for learnability of data-driven models for CPS exist?*.

Table II — Concepts and keywords for a systematic literature review on learnability of data-driven models for CPSs.

Concept	Keywords
Cyber-Physical Systems	CPS, Cyber-Physical System, Hybrid System, Hybrid Automaton, Mixed System, Dynamical System, Industrial System
Learnability	Data-Driven, Machine Learning, System Identification, Model Learning
Data-Driven Methods	Learnability, Consistency, Optimality, Guarantee, Trustworthiness, Faithfulness, Conformal Prediction, Robustness

Table III — Excluded concepts and keywords for a systematic literature review on learnability of data-driven models for CPSs.

Concept	Keywords
Control	Control, Lyapunov
Model Types	Manual, LLM
Applications	Communication, Privacy
Modeling Goals	Real-Time, Attack, Security

Based on this research, we identify a set of relevant concepts and keywords, which we use to construct a search query. The used concepts are *Cyber-Physical Systems*, *Learnability*, and *Data-Driven Methods*. The keywords derived from these concepts are shown in Table II.

Further, we identify explicit exclusion subjects leading to the exclusion keywords in Table III. The exclusion keywords are used to filter out papers that are not relevant to the research question. Those are papers that focus on control, as we are not interested in the learning of controllers, but in the learning of models that represent an abstraction of a system. We also exclude papers that focus on specific model types, such as Large Language Model (LLM), as papers discussing LLMs often do not cover the model learning, but rather the application of the model. Also, we exclude manually designed models, as we are interested in data-driven models. Further, we exclude papers that focus on communication and security applications. These subjects are part of the initial test queries as learnability terms such as trustworthiness and consistency are relevant keywords in these applications, though, not in the context of learnability. Finally, we exclude papers that focus on real-time applications, attacks, and privacy. Again, test queries showed an overlap in keywords with these fields, but the usage of these terms is not in the context of learnability.

With these concepts and exclusion terms, we form the following search query to match with the topic (abstract, title, and keywords) or all fields of the papers.

Topic **contains**	"Guarantee" OR "Optimality" OR "Trustworthiness" OR "Learnability" OR "Faithfulness" OR "Conformal Prediction" OR "Robust Learning" OR "Learner Robustness"
AND All **contains**	"Learned Model" OR "Model Learning" OR "System Identification" OR "Data-driven"
And All **contains**	"Hybrid System" OR "Dynamical System" OR "Mixed System" OR "Hybrid Automaton" OR "Hybrid Automata" OR "Cyber-Physical Systems"
AND Topic **contains**	"Data-driven" OR "Learned Model" OR "Model Learning" OR "Identification"
AND Topic **contains**	"Cyber-Physical System" OR "Industrial" OR "Hybrid Automaton" OR "Hybrid Automata"
AND NOT Topic **contains**	"Security" OR "Communication" OR "Attack" OR "Large-Language Models" OR "LLM" OR "Privacy" OR "Control" OR "Real-Time"
AND NOT all **contains**	"Lyapunov" OR "Ljapunow" OR "Ljapunov" OR "Ljapunoff" OR "Liapunov".

We use the search query to search for relevant papers in the databases and repositories arXiv, IEEE Xplore, Scopus, and Web of Science. The search finds 38 papers, from which we exclude 21 papers through manual inspection, resulting in 17 relevant papers.

2.2.2 Spectrum of the Literature with respect to Data-Driven Modeling

The spectrum from data-driven methods automatically generating models to manual model design from prior knowledge is a core theme in the reviewed papers. Such prior knowledge often exists beyond the diversity of CPS and application domains and derives, for instance, from the underlying phenomena in the physical components of the system.

Figure 8 illustrates this spectrum, showcasing a selection of papers from our review that exemplify different points along the spectrum. At one end of the spectrum, models are derived almost entirely through domain expertise and manual design. J. Jeon et al. [83] emphasize the importance of constructing trustworthy models through manual efforts, demonstrating how domain knowledge and modeling expertise play an important role in ensuring the reliability of manually developed models.

As we move along the spectrum, models begin to integrate data-driven elements, such as accounting for uncertainty. X. Xin et al. [84] illustrate this approach by incor-

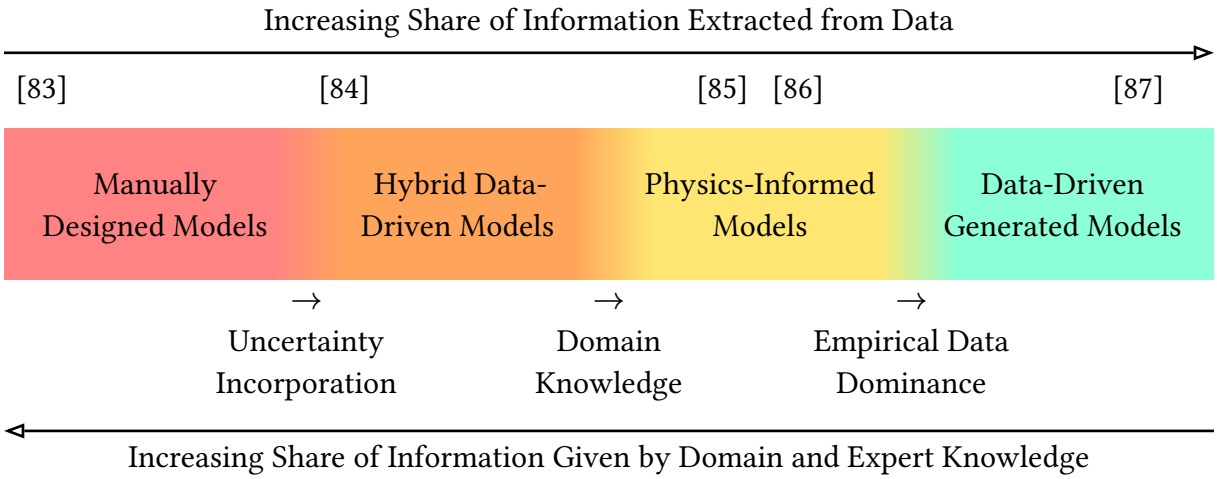


Figure 8 — Spectrum of data-driven influence in modeling of CPSs [82] with respect to domain and expert knowledge. The figure illustrates the varying degrees of reliance on data-driven methods. On the left end, models are derived almost entirely through domain expertise and manual design. The right end represents models that are generated predominantly from data-driven approaches. Hybrid and physics-informed methods occupy the middle ground, leveraging both data and domain knowledge.

porating pre-recorded sensor data to update an automaton's transition probability matrix. This method addresses sensor uncertainty, improving upon the industrial state-of-the-art that assumes flawless sensor functionality. By acknowledging and adapting to real-world data imperfections, the model reflects a more accurate understanding of system behavior.

Further along the spectrum lie physics-informed or domain knowledge-informed methodologies. These approaches blend theoretical insights with data-driven modeling to enhance model accuracy. For instance, Z. Wang et al. [85] integrate physics-based information into neural network training, which enhances the model's fidelity by grounding data-driven processes in established scientific principles. Additionally, H. Wang et al. [86] propose a framework to seamlessly incorporate domain knowledge into data-driven modeling efforts, thereby enriching the process with valuable prior knowledge and improving the resulting model performance.

At the opposite end of the spectrum lies purely data-driven methodologies, wherein system behavior is derived solely from empirical data. These models utilize advanced data analytics and machine learning algorithms to create dynamic and adaptable representations that potentially uncover complex system dynamics difficult to capture with traditional approaches. G. Dang et al. [87] exemplify this approach by introducing a recurrent stochastic configuration network to learn complex nonlinear systems.

Ultimately, this spectrum illustrates the diverse approaches to CPS modeling, highlighting hybrid methodologies that leverage both expert knowledge and data-driven insights to enhance model reliability and accuracy. Due to the multifaceted nature

of CPS, no single technique prevails as superior. The choice of modeling approach depends on specific system requirements and desired model properties as well as the available domain knowledge and data.

2.2.3 Learnability in the Learning Process of Cyber-Physical Systems

The section presents a structured view of the literature considering four key steps in the data-driven modeling process for CPS:

- data quality assurance,
- model learning,
- model evaluation, and
- model refinement.

Each step is influenced by the integration of prior knowledge, which can enhance data generation, guide learning algorithms, provide evaluation benchmarks, and identify areas for model improvement.

Ensuring high-quality data is foundational for model learnability. Y. Zeng et al. [88] propose methods to identify under-represented data regions and points near classification boundaries to efficiently generate informative samples. J. Jeon et al. [83] use digital twins to generate synthetic data for marine engine models to improve model performance in fault detection, identification, or isolation. F. Su et al. [89] enhance feature extraction for fault diagnosis by integrating wavelet transforms into convolutional neural networks.

The step of actual model learning covers a spectrum of approaches, from purely data-driven to hybrid and knowledge-informed methods. G. Dang et al. [87] introduce recurrent stochastic configuration networks for learning complex non-linear system dynamics. G. Saveri et al. [90] mine STL formulae from time-series data using Bayesian optimization and information retrieval. G. Chen et al. [91] and A. Brusaferri et al. [92] focus on learning and abstracting piecewise affine hybrid systems, with the latter incorporating neural networks for probabilistic transition guards called boundaries. Silvia Maria Zanoli et al. [93] advocate for learning multiple hybrid automata to better capture system behavior. M. Waga et al. [94] leverage reinforcement learning for learning FSMs with dynamic shielding, which prevents the system from executing unsafe actions. This method is applied to black-box systems with unknown dynamics. Other works, such as X. Xin et al. [84], H. Wang et al. [86], and Z. Wang et al. [85], demonstrate the value of incorporating prior knowledge to improve learnability and robustness through expert knowledge. These works cover the integration of knowledge on sensor functionality in parameter identification of neural networks, the combination of physics-based equations, and the training of neural networks with physics-based loss functions, respectively.

Evaluation metrics are crucial for assessing learnability. As mentioned, Z. Wang et al. [85] use physics-informed loss functions to align models with physical

laws. A. Devonport et al. [95] introduce PAC bounds for data-driven reachability analysis, providing probabilistic guarantees of model accuracy. C. Sun et al. [96] define robustness as a metric and propose strategies to improve model performance under domain shifts. J. Jeon et al. [83] further contribute with a comprehensive evaluation framework including validation, verification, and robustness analysis.

Model refinement is an iterative process triggered by evaluation results. F. Vitale et al. [97] refine Petri net models for digital twins through offline and online learning. G. Prasad et al. [98] present an online algorithm for refining self-organizing fuzzy neural networks, balancing generalization, coverage of the input space, and model size. C. Sun et al. [96] distinguish refinements in model structure and model parameters, proposing calibration strategies to address the former and suggesting further refinement for the latter.

2.3 Summary

Concluding, there is a wide range of approaches for learning models of CPSs. These approaches range from classical manual modeling over data-driven machine learning and automata learning and its variations, e.g., hybrid and timed automata, to the use of probabilistic models. The learnability of CPS models is a multifaceted challenge addressed through a combination of data quality assurance, advanced learning algorithms, rigorous evaluation, and iterative refinement, with prior knowledge playing a pivotal role throughout the process. The reviewed literature demonstrates a rich interplay between data-driven and knowledge-based approaches, highlighting the need for adaptable and robust modeling pipelines in the CPS domain. Here, we focus on the learning of interpretable and compact models, such as DFAs, Mealy machines, and hybrid automata and address the challenges in learning such interpretable models from data.

Chapter 3

Preliminaries

This section introduces the basic concepts and definitions. We introduce the models and learning methods used in this thesis. First, we present decision trees and their learning methods in Section 3.1. Next, we present the different types of automata, i.e., DFAs and Mealy machines, and introduce the concept of active and passive automata learning in Section 3.2. Finally, Section 3.3 introduces hybrid automata as an extension of discrete automata. Section 3.4 provides a brief formal introduction to systems and observations. Further, we present methods, such as Dynamic Time Warping (DTW) and Symbolic Regression, used in the proposed approaches in Section 3.5 and Section 3.6. We conclude this section with a description of the examples used throughout this thesis in Section 3.7.

3.1 Decision Trees

Decision trees are an established method for classification and regression tasks. Further, they are discrete and deterministic models. A particular benefit of decision trees is their interpretability. They are easy to understand and visualize, making them suitable for applications where interpretability is important.

Definition 3.1: A *decision tree* [99], [100] is a tuple $\mathcal{T} = (V, E)$ with nodes or vertices V and edges E. The nodes and edges form a tree, i.e., a connected, acyclic graph. The tree represents a classifier $d : \mathbb{F} \to C$. The set $\mathbb{F}$ forms a space of vectors $\boldsymbol{f}$, where $\boldsymbol{f} = [f_1, ..., f_m]$ holds feature values f_i from feature spaces $\mathbb{F}_i$ and C is a set of classes.

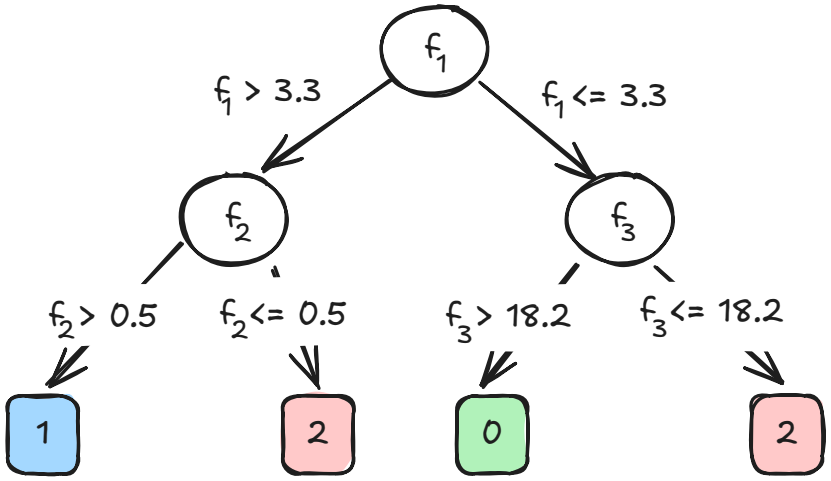


Figure 9 — Example of a decision tree. The tree is used to classify observations based on the values of the features. The inner nodes and edges are associated with features and split conditions, while the leaves are associated with class labels.

The tree $\mathcal{T}$ has exactly one node without incoming edges that is the root r. The set of nodes with no outgoing edges are the leaves V_L of the tree. All nodes $V \setminus V_L$ are *inner nodes* of the tree. Each node v in $\mathcal{T}$ represents a subset $\mathcal{O}_v$ of $\mathbb{F} \times C$. The root r represents the entire set of observations $\mathbb{F} \times C$. Each edge is associated to a feature f_i and a split condition $t_i : \mathbb{F}_i \to \{1, ..., n\}$ on this feature, where n is the number of children of a considered leaf v. The set $\mathcal{O}_v$ represented by node v is split by the condition into subsets $\mathcal{O}_{v_1}, ..., \mathcal{O}_{v_n}$. For binary decision trees, we have $n = 2$. The children of v represent the subsets $\mathcal{O}_{v_1}, \mathcal{O}_{v_2}, ..., \mathcal{O}_{v_n}$. Each leaf l has a label $c \in C$ given by the class with the highest frequency in the corresponding subset:

$$c_l = \underset{\tilde{c} \in C}{\operatorname{argmax}} \, |\{\langle \boldsymbol{f}, \hat{c} \rangle \in \mathcal{O}_l : \hat{c} \equiv \tilde{c}\}|. \tag{1}$$

Every new feature vector $\boldsymbol{f}$ is classified by traversing the tree from the root to a leaf based on the feature values and the split conditions. The result of the classification is the label of the leaf reached by the traversal.

Example 1: Figure 9 shows an example of a decision tree. The tree has three features f_1, f_2, and f_3 and three classes $C = \{1, 2, 3\}$. The inner nodes and edges are associated with features and split conditions, while the leaves are associated with class labels.

3.1.1 Decision Tree Learning

For DTL, a set of labeled observations is given, where each observation consists of a feature vector $\boldsymbol{f}$ and a class label $c \in C$. The goal of the learning process is to find a decision tree $\mathcal{T}$ that classifies the observations correctly. Learning algorithms for

decision trees use different splitting criteria to determine the best feature and split condition for each inner node of the tree [20]. The splitting criteria are based, e.g., on the information gain or impurity of the subsets created by the split. The most common splitting criteria are the Gini impurity and the entropy [101]. During the learning process, the tree is recursively split until all leaves are pure, i.e., originate from the same class, or a stopping criterion is met. The stopping criterion can be based on the maximum depth of the tree, the minimum number of samples in a leaf, or the minimum impurity decrease.

3.1.2 Regression Trees

A variant of decision trees is the regression tree. Regression trees are used for regression tasks, where the goal is to predict a continuous value based on the input features.

Definition 3.2: A *regression tree* [20] is a decision tree where the leaves are associated with real-valued outputs.

Here, the label of a leaf l is the average of the target values of the observations in the subset $\mathcal{O}_l$ associated with the leaf, i.e., the decision function d of the regression tree is defined as

$$d(\boldsymbol{f}) = \frac{1}{|S_l|} \sum_{\langle \tilde{\boldsymbol{f}}, y \rangle \in \mathcal{O}_{\boldsymbol{f}}} y, \tag{2}$$

where $\mathcal{O}_{\boldsymbol{f}}$ is the set of observations associated with the leaf that is reached with the feature vector $\boldsymbol{f}$.

Regression Tree Learning (RTL) is similar to DTL, but the splitting criteria are based on the continuous outputs. Here, we use a splitting criterion that minimizes the Mean Squared Error (MSE) of the target values in the subsets created by the split.

3.2 Finite State Machines

Finite state machines are a class of discrete models that are used to represent the behavior of systems or processes. They consist of a finite number of states, transitions between these states, and an alphabet of symbols that describe the behavior of the system.

Definition 3.3: A *Finite State Machine (FSM)* [102] is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \theta)$ where Q is the set of states, Σ is the alphabet, q_0 is the initial state, and $\theta : Q \times \Sigma \to Q^P$ is the transition function, where Q^P is the power set of Q. The transition function defines how the system transitions between states and labels transitions with symbols from the alphabet.

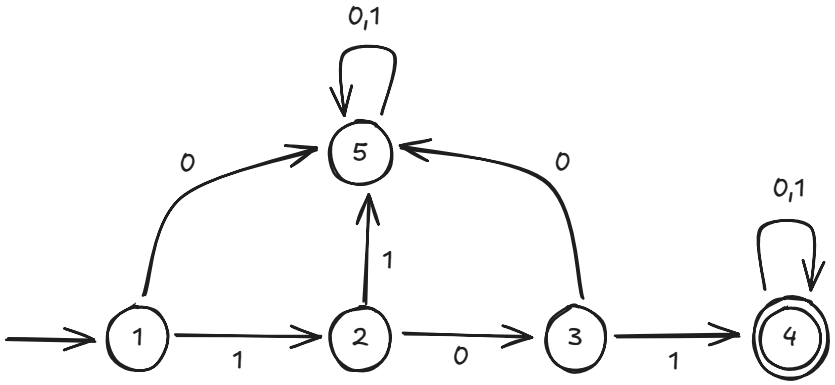


Figure 10 — Example of a DFA with five states and the input alphabet $\Sigma = \{0, 1\}$. State 1 is the initial state and State 4 is an accepting state of the automaton.

There are different types of finite state machines, where the two used in this work are Deterministic Finite Automata (DFAs) and Mealy machines. Note that we use some of the notation, e.g., for the set of states Q and the alphabet Σ, for all types of FSMs.

Definition 3.4: A *Deterministic Finite Automaton (DFA)* is a tuple $\mathcal{A}_{\text{DFA}} = (\Sigma, Q, q_0, \theta, Q_F)$, with Σ being the input alphabet. The finite, non-empty set Q is the set of states and q_0 is an initial state with $q_0 \in Q$. The set Q_F is the set of final states with $Q_F \subseteq Q$. The state transition function θ is defined as $\theta : Q \times \Sigma \to Q$.

A DFA [102] is a finite state machine that accepts or rejects input strings based on the current state and the input symbol. The machine starts in the initial state q_0 and processes the input string symbol by symbol. A DFA represents a language $L \subseteq \Sigma^*$. Here, Σ^* denotes the Kleene closure of Σ, i.e., the set of traces from symbols of Σ of arbitrary finite length. A trace $\boldsymbol{s} = [s_1, ..., s_n]$ with $s_i \in \Sigma$ is in L if and only if $s_1 = q_0$ and there is a run on $\mathcal{A}_{\text{DFA}}$ where the trace $[s_1, ..., s_n]$ leads to a final state $q_F \in Q_F$. We denote the inference of the DFA by using the automaton as a function $\mathcal{A}_{\text{DFA}}(\boldsymbol{s}) \in \{0, 1\}$, where $\boldsymbol{s} \in \Sigma^*$ is the input trace that is accepted by the automaton if the result is 1 and rejected if the result is 0.

Example 2: Figure 10 shows an example of a DFA with five states, where the State 4 is the accepting state. The automaton accepts the language $L = \{101 \cdot^*\}$, where $\cdot^*$ denotes any sequence from the alphabet $\Sigma = \{0, 1\}$ (including the empty sequence).

Definition 3.5: A *Mealy machine* [68] is a tuple $\mathcal{A}_{\text{M}} = (Q, I, O, \alpha, \beta, q_0)$ where Q is the set of states with $q_0 \in Q$ being the initial state. I and O are the input and output alphabets, respectively. The functions α and β are the output and transition functions, respectively.

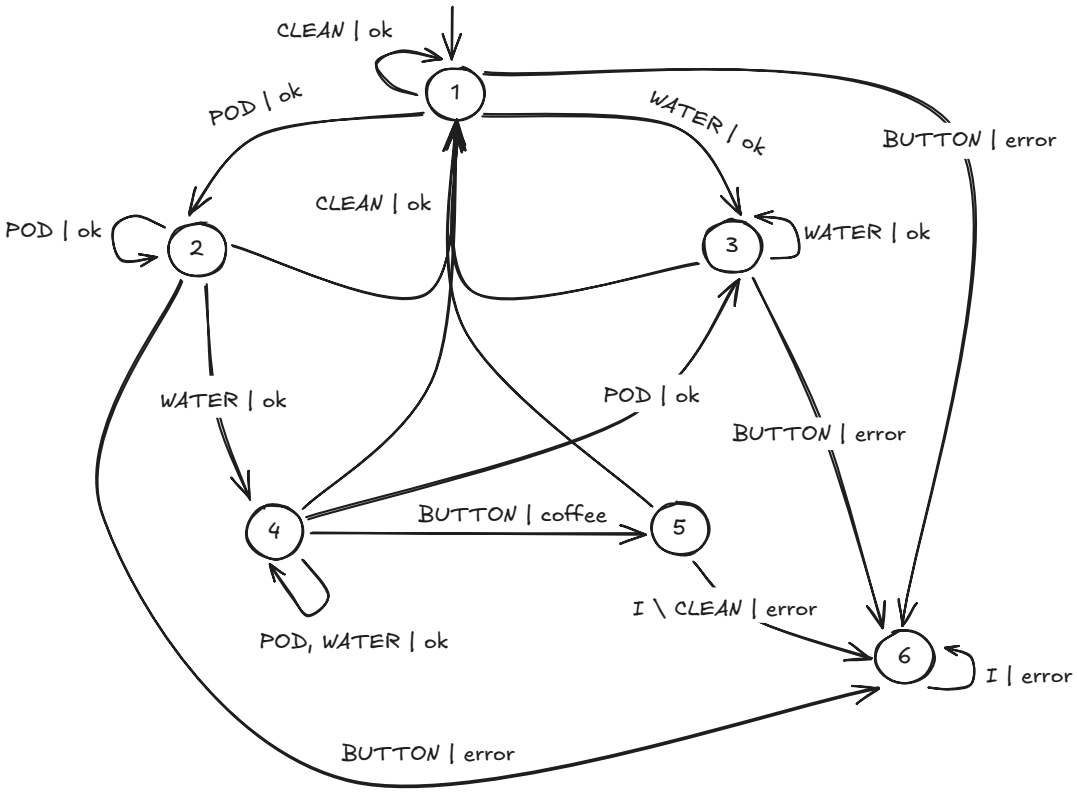


Figure 11 — Mealy machine representation of a coffee machine. The input alphabet is $I = \{\texttt{BUTTON}, \texttt{CLEAN}, \texttt{POD}, \texttt{WATER}\}$ and the output alphabet is $O = \{\texttt{ok}, \texttt{error}, \texttt{coffee}\}$.

A Mealy machine represents a language $L \subseteq \Sigma^*$ on the alphabet $\Sigma = I \times O$ equivalent to DFAs. A trace $s = [(i_1, o_1), ..., (i_n, o_n)] \in \Sigma^*$ is in L if and only if there is a run on M where the input trace $[i_1, ..., i_n]$ produces the output $[o_1, ..., o_n]$. We denote the inference of the Mealy machine by using the automaton as a function $\mathcal{A}_{\mathrm{M}}(\boldsymbol{i}) \in O^{|\boldsymbol{i}|}$, where $\boldsymbol{i} \in I^*$ is the input trace that produces the output trace $[o_1, ..., o_{|\boldsymbol{i}|}]$.

Example 3: In Figure 11, an example of a Mealy machine is shown. The example represents a coffee machine with the input alphabet $I = \{\texttt{BUTTON}, \texttt{CLEAN}, \texttt{POD}, \texttt{WATER}\}$ and the output alphabet $O = \{\texttt{ok}, \texttt{error}, \texttt{coffee}\}$. The language of the automaton is

$$
\begin{aligned}
L = \{&\texttt{CLEAN|ok},\\
&\texttt{CLEAN|ok, POD|ok},\\
&...,\\
&\texttt{POD|ok},\\
&\texttt{POD|ok, POD|ok},\\
&\texttt{POD|ok, CLEAN|ok},\\
&...\}
\end{aligned}
\qquad (3)
$$

```
1:  function RPNI(O_+, O_-)
2:      A ← buildPTA(O_+)
3:      Q_red ← {q_0}
4:      Q_blue ← {θ(q_0, σ) : σ ∈ Σ}
5:      while Q_blue ≠ ∅ do
6:          q ← Q_blue.pop()
7:          if ∃q_r ∈ Q_red with compatibleMerge(A, q_r, q_b, O_-) then
8:              Q_red ← merge(A, q_r, q_b)
9:          end
10:         else
11:             Q_red ← Q_red ∪ {q_b}
12:         end
13:         Q_blue ← Q_blue ∪ {θ(q, σ) : σ ∈ Σ}
14:     end
15:     return A
16: end
```

Listing 1 — Regular Positive-Negative Inference (RPNI) algorithm for passive learning of a DFA from positive and negative observations [79]. The algorithm uses the positive observations to create a tree structure and successively merges the tree nodes into an automaton considering the negative observations. The result is a DFA that accepts the positive observations and rejects the negative observations.

The data-driven identification of FSMs, also termed *grammatical inference*, differentiates two major paradigms, passive and active automata learning, which we explore in the following sections.

3.2.1 Passive Automata Learning

Passive automata learning takes a number of observations from a system and learns an automaton on these observations. There is no direct connection between the learner and the System Under Learning (SUL), but observations are collected prior to learning. For learning of DFAs, a learning set has to contain positive observations $\mathcal{O}_+$ and negative observations $\mathcal{O}_-$, where positive observations are traces that are accepted by the SUL and negative observations are traces that are not accepted by the SUL.

A common learning algorithm for DFAs is the RPNI algorithm [103]. The RPNI algorithm is described in Listing 1 and implements the following steps:

- The algorithm constructs a Prefix Tree Acceptor (PTA) from the positive observations $\mathcal{O}_+$ in Line 2. The PTA represents the positive observations in a tree structure,

whose root node is the initial state of the DFA and all positive observations are represented as paths in the tree from the root node to the leaf nodes.
- The algorithm holds two sets of states named as red states Q_{red} and blue states Q_{blue}, initialized in Line 3 and Line 4 of Listing 1. The red states are the states that are already determined to be part of the DFA and the blue states are candidates for merging with the red states.
- The algorithm iteratively takes a random blue state q in Line 6 and checks if it is compatible with any red state q_r in Line 7. The states are compatible, if a merge of the two states does not accept any negative observation.
- If compatible states are found, they are merged in Line 8. Otherwise, the blue state is added to the set of red states in Line 11.
- All successors of q are added to the set of blue states in Line 13 of Listing 1.
- The algorithm continues until no blue states are left, i.e., all states are states of the DFA according to the stopping criterion of the loop in Line 5.
- The resulting DFA is returned as the output of the algorithm in Line 15.

For Mealy machines, the learning process is similar to the one for DFAs. Nevertheless, for Mealy machines, no negative observations are needed. The learning process is based on the positive observations and the output of the system. Two states in the PTA are compatible for a merge if the output for the same input symbol is the same for both states [79].

The RPNI algorithm identifies a target DFA or a Mealy machine in the limit, i.e., there exists a *characteristic set*, for which the learning algorithm identifies the target automaton [79]. Adding new observations to the characteristic set does not change the learned automaton. Learning the target automaton on a characteristic set is polynomial in time and data with respect to the sum of the lengths of all traces in the learning set [104], [105].

3.2.2 Active Automata Learning

During active automata learning, a hypothesis model M is gradually refined by using membership queries and comparing it to the SUL using equivalence queries as shown in Figure 12 [106]. If the equivalence oracle discovers diverging behavior between the model and the system, a counterexample is created and returned to the learner. The hypothesis model is refined using membership queries that are derived from the counterexample. As soon as no new information results from the counterexample, the learner again asks the equivalence oracle until it finds another counterexample or the equivalence oracle determines the model of the system is correct.

Complete equivalence oracles are able to determine whether the model is equivalent to the SUL [68]. Usually, such equivalence oracles do not exist especially for complex and black-box systems. Instead, approximate equivalence oracles are used to determine whether the model is equivalent to the SUL within a certain threshold. One example for an equivalence oracle is a random equivalence oracle, which generates up to $j \in \mathbb{N}$

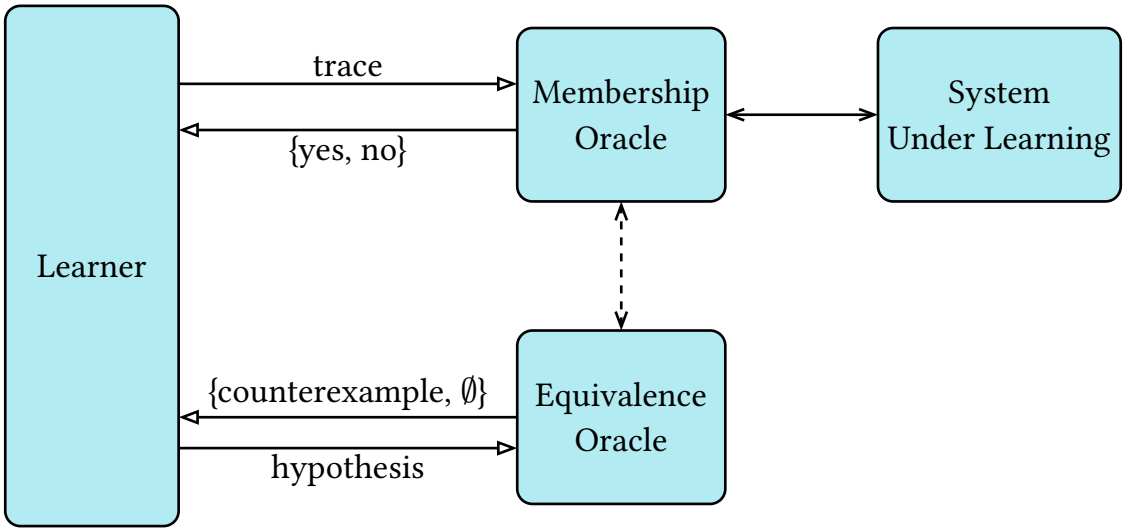


Figure 12 — Active automata learning process. The learner refines the model using membership queries and equivalence queries [106].

random input traces with a length $l \in \mathbb{N}$ from the input alphabet and uses them to compare the learned model against the system. If the behavior of the model and the system deviate from one another for one of the random traces, this trace is returned as the counterexample. The number j of generated random traces is the threshold by which the equivalence oracle decides when the learning process is stopped and the model is considered accurate enough. The number j of traces and their length l are adjustable, which has a direct influence on the quality of the model.

A well known example for an active automata learning algorithm, which implements the procedure of Figure 12, is the L*-algorithm [67]. Convergence in the limit is guaranteed for the L*-algorithm, i.e., the algorithm converges to the target automaton as soon as a characteristic set is reached. For a complete equivalence oracle, the L*-algorithm is guaranteed to converge in polynomial time and data with respect to the size of a characteristic set of the system under learning. The characteristic set, again, is polynomial in the size of the automaton [104].

3.3 Hybrid Automata

We use hybrid automata as an extension of FSMs that combine discrete and continuous behavior.

Definition 3.6: A hybrid automaton is a 5-tuple $\mathcal{A}_H = (X, Q, F, K, T)$ where

- $X = \{x_0, x_1, x_2, ..., x_n\} = I \cup O \cup S$ is the set of system variables, which consist of input variables I, output variables O and state variables S. The state variables may include the continuous time t. Also, derivatives of variables may form individual variables in X.
- Q is the set of modes.

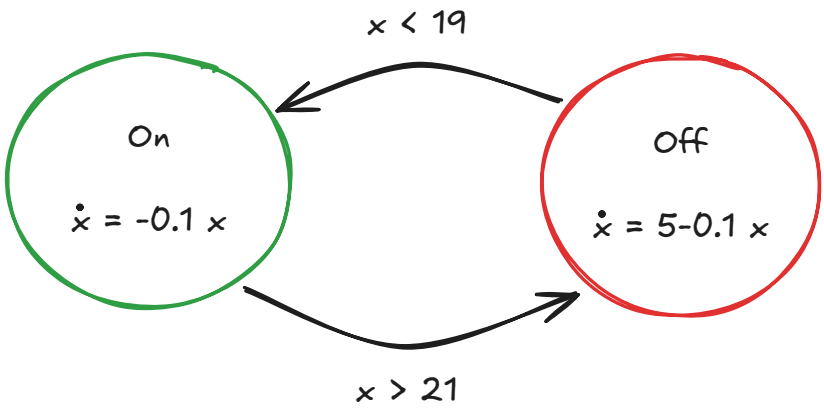


Figure 13 — Hybrid automaton representation of a boiler system. The system has two modes: the On mode (heating) and the Off mode (cooling), which are described by differential equations [107].

- F is the set of flow functions. A flow function $f_q \in F$ defines the change of the state variables S as well as the output variables O within the mode $q \in Q$.
- K is a set of guards leading to transitions between the modes. Each guard is a set of conditions on the variables in X.
- $T : Q \times K \to Q$ defines transitions between the modes. A transition happens if the corresponding guard $k \in K$ is active.

The discrete behavior is captured by the discrete modes Q and the guards K. The continuous dynamics are represented by the flow functions F of the modes.

Example 4: Figure 13 shows an example of a hybrid automaton representing a boiler system. The system has two modes $Q = \{\text{On}, \text{Off}\}$, the state variables are $X = \{x, \dot{x}\}$. The flow functions are given as differential equations for the two modes with $f_{\text{On}} : \dot{x} = -0.1x$ and $f_{\text{Off}} : \dot{x} = 5 - 0.1x$. The guard condition for switching from the *on* mode to the *off* mode is $g_{\text{On}} = \{x > 21\}$ and the guard condition for switching from the *off* mode to the *on* mode is $g_{\text{Off}} = \{x < 19\}$.

3.4 System & Observations

In the context of data-driven learning for CPSs, observations are collected from the SUL to form a learning set. For the systematic description of system behavior and data collection, we differentiate the following terms:

- *run*: A run is a single execution or simulation of the system. Runs differ in the initial conditions, the input signals, the parameterization, or the environment.

- *sample*: A sample is a single snapshot of the system variables, i.e., state, input, and output variables at a specific point in time. Samples are discrete symbols or continuous values or vectors of these in a multidimensional state space.
- *trace*: A trace $\boldsymbol{s}$ is a sequence of samples of the system from a run. The trace consists of sub-traces for the individual input signals $\boldsymbol{i_1}, \boldsymbol{i_2}, ...$, output signals $\boldsymbol{o_1}, \boldsymbol{o_2}, ...$, and potentially inner state variables $\boldsymbol{x_1}, \boldsymbol{x_2}, ...$ of the system, i.e.,

$$\boldsymbol{s} = \begin{bmatrix} \boldsymbol{i_1} \\ \boldsymbol{i_2} \\ \vdots \\ \boldsymbol{o_1} \\ \boldsymbol{o_2} \\ \vdots \\ \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \vdots \end{bmatrix}. \tag{4}$$

- *observation*: An observation is a single element of the learning set $\mathcal{O}$. Observations are generated from traces, but depending on the type of model or learner, the traces are processed or transformed into a specific format to serve as suitable observations.

For traces from hybrid automata, we define *transition points* as the last sample in a trace before a transition between modes of the model occurs.

3.5 Dynamic Time Warping

Dynamic Time Warping (DTW) [108] is a technique to compare two traces $\boldsymbol{a}$ and $\boldsymbol{b}$ of length k and h, respectively, that may vary in speed or sampling rate. The result of the comparison is a similarity measure for the two traces.

The comparison with DTW identifies an alignment path $p = [p_1, ..., p_L]$ of length L for the two sequences $\boldsymbol{a}$ and $\boldsymbol{b}$. An element $p_l = \langle r_l, s_l \rangle$ in the alignment path represents a mapping between elements $\boldsymbol{a}[r_l]$ and $\boldsymbol{b}[s_l]$ in the two traces:

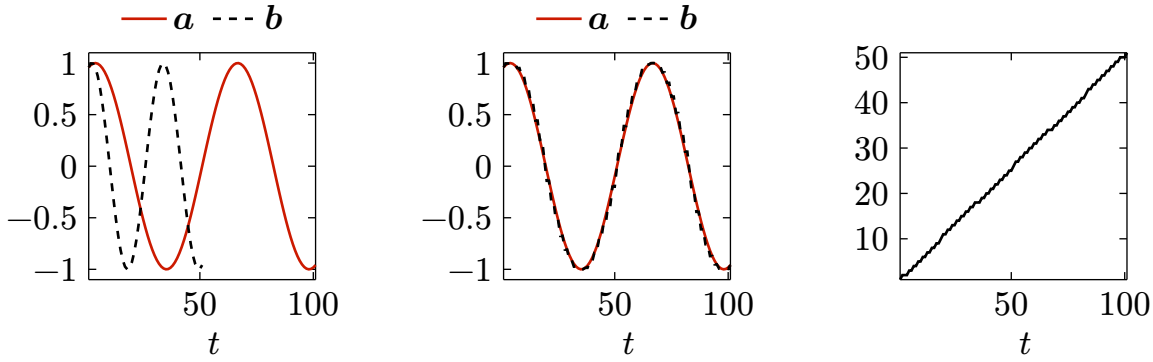
$$p_l = \langle r_l, s_l \rangle \in \{1, ..., k\} \times \{1, ..., h\}, l \in \{1, ..., L\}. \tag{5}$$

For the construction of the alignment path, the following conditions hold [108]:

- boundary condition: $p_1 = \langle 1, 1 \rangle$ and $p_L = \langle k, h \rangle$,
- monotonicity: $r_1 \leq r_2 ... \leq r_L$ and $s_1 \leq s_2 ... \leq s_L$, and
- step size: $p_{l+1} - p_l \in \{\langle 1, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 1 \rangle\}, \quad \forall l \in [1, ..., L]$.

The alignment path maps similar elements from the two traces by minimizing the sum of absolute differences

$$\zeta_{\text{dist}}(\boldsymbol{a}, \boldsymbol{b}) = \sum_{l=1}^{L} |\boldsymbol{a}[r_l] - \boldsymbol{b}[s_l]| \tag{6}$$



(a) Sine with two different sampling periods.

(b) Aligned sines using the DTW alignment path.

(c) Alignment path of the two sines.

Figure 14 — DTW comparison of two sines. DTW aligns the two signals despite their different sampling rates and lengths.

between the mapped elements. The implementation of DTW comparisons uses a greedy search to find an alignment path that minimizes the distance ζ_{dist}.

Two signals are considered similar if the distance, as defined in Equation 6, is small, and the alignment path is diagonal [109]. The diagonality metric $\zeta_{\text{diag}}([r_1, r_2, ..., r_L], [s_1, s_2, ..., s_L])$ is the correlation coefficient of the mapped indices in the two traces. Finally, the distance metric and diagonality metric are combined into a single metric as proposed in [109] by a weighted sum as follows

$$\zeta_{\text{DTW}}(\boldsymbol{a}, \boldsymbol{b}) = w_{\text{diag}} \cdot \zeta_{\text{diag}}([r_1, r_2, ..., r_L], [s_1, s_2, ..., s_L]) + w_{\text{dist}} \frac{1}{L} \cdot \zeta_{\text{dist}}(\boldsymbol{a}, \boldsymbol{b}), \quad (7)$$

where $w_{\text{diag}} + w_{\text{dist}} = 1$ holds.

> *Example 5*: In Figure 14, the results of a DTW comparison of a sine sampled with two different sampling periods are displayed. The distance of the two signals is low, and the alignment path is diagonal. Thus, the two signals are considered similar.

Example 5 shows that the DTW comparison is robust against variations in the sampling rate and the length of the two compared traces.

3.6 Symbolic Regression

Symbolic Regression (SR) is a machine learning technique that finds symbolic expressions for the relationship between input and output variables. The goal is to find a function r that maps input features $\boldsymbol{f}$ to an output variable o such that $o = r(\boldsymbol{f})$. The function r is represented as a symbolic expression in terms of a set B of basic functions and operators applied to the input variables [110]. The search for the best symbolic expression is guided by a fitness function that evaluates the quality of candidate symbolic expressions on a learning set. Often, a trade-off between the accuracy of the expression on the learning data and the length of the candidate expression is used to evaluate the fitness of the expression. This strategy avoids a bloat of the expressions [111]. A

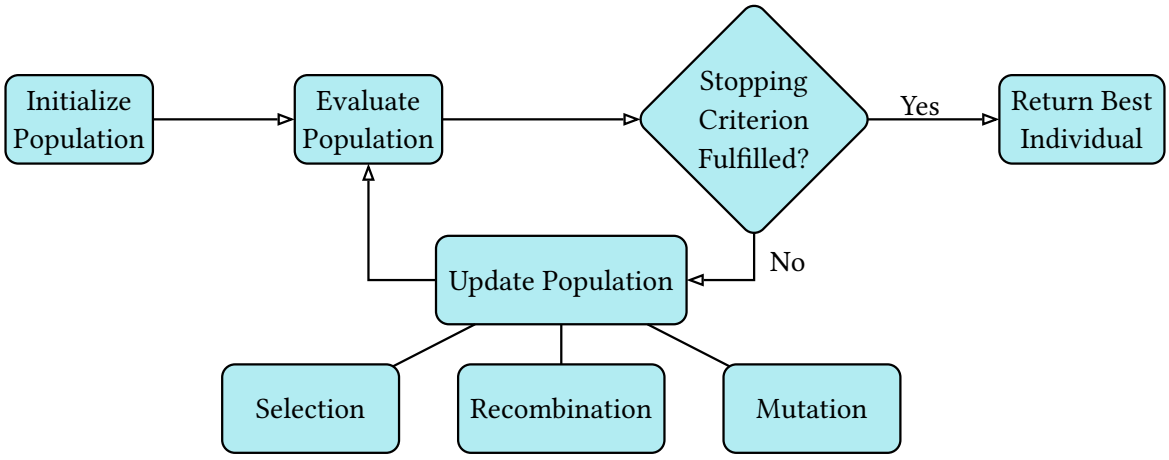


Figure 15 — Genetic programming algorithm for symbolic regression. The algorithm starts with a population of random expressions and iteratively refines them using genetic operators such as selection, mutation, and recombination.

parsimony coefficient ψ works as a regularization parameter in this multi-objective optimization problem.

In the scope of this thesis, we use symbolic regression via genetic programming [110]. Genetic programming performs the search for the best symbolic expression by evolving a population of candidate expressions over so-called *generations*. Given a learning set $\mathcal{O}_{\text{SR}}$ containing observations of the output variable o and the input features $\boldsymbol{f}$, the symbolic regression algorithm learns a symbolic expression r. We denote the symbolic regression algorithm as a function which returns a symbolic expression r as well as the remaining error e of the symbolic expression on the learning set $\mathcal{O}_{\text{SR}}$ after η generations:

$$r, e = \textsc{learnExpression}(\mathcal{O}_{\text{SR}}, \eta). \tag{8}$$

The remaining error e of the symbolic expression r is evaluated on the learning set $\mathcal{O}_{\text{SR}}$. A potential error metric is the MSE of the symbolic expression on the learning set. Figure 15 illustrates the learning process. A population of size p is initialized with random expressions. In each generation, the population is evaluated using a fitness function. New population of candidate solutions is updated by applying genetic operators, such as

- selection, which selects a subset of the candidates for the population of the next generation. Usually, the best candidates are selected based on their fitness scores. Further,
- mutation introduces changes in the structure of the candidates, e.g., adding or deleting nodes or swapping an operator, and
- recombination combines parts of two or more candidate solutions to create new ones.

The process continues until a stopping criterion is fulfilled, e.g., a predefined number η of generations is reached.

Table IV — Parameters for the symbolic regression algorithm.

Parameter	Symbol	Description
Population size	p	The number of candidate expressions in the population.
Number of generations	η	The number of generations to evolve the population.
Parsimony coefficient	ψ	The regularization parameter that controls the trade-off between the error and the length of the expression.
Basic functions	B	The set of basic functions and operators used to construct the symbolic expressions.

Symbolic regression is not guaranteed to find the original target function even under complete and noise-free data [112]. The parameterization of the symbolic regression algorithm has a significant impact on the quality of the learned symbolic expression. Table IV shows the parameters used in Chapter 6. A larger population size p increases the diversity of the candidates and can lead to better solutions, but also increases the computational cost. The number of generations n determines how long the algorithm runs and how many candidates are evaluated. A larger number of generations can lead to better solutions, but again increases the computational cost. A population can be restored from a checkpoint to continue the evolutionary process to, e.g., reduce the error of a candidate solution through additional generations.

The parsimony coefficient ψ controls the trade-off between the error and the length of the expression. This is formulated, e.g., in an effective error $\bar{e}$, which weights the complexity of an expression with its accuracy:

$$\bar{e} = e + \psi \cdot |r| \tag{9}$$

where $|r|$ is the length of the expression r and e is, again, the remaining error of the expression on the learning set. This trade-off prevents bloat of the expressions [111], [113]. The search space of the symbolic regression algorithm is defined by the set of functions over the basic functions in the set B and further controlled by the parsimony coefficient ψ, which penalizes more complex expressions. The larger the search space, the more complex is the learning problem. With a larger search space the identification of the target function is less likely. Apart from the parameterization, the amount of data and the range of captured behavior are crucial for the success of the algorithm.

3.7 Examples

Throughout this thesis, we use several examples and case studies as listed in Table V. These systems illustrate conceptual and theoretical results. Furthermore, the systems are used to evaluate the proposed methods and algorithms. Thus, the examples appear

Table V — Overview of the examples and their appearances this thesis.

Example	Sections
Coffee machine	Section 4.5, 5.2, 5.4
Water tank	Section 5.2, 5.3, 7.3
Two tank system	Section 6.3, 6.4
Boiler	Section 5.2, 5.3, 6.3, 6.4
Power converter	Section 6.3, 6.4
Vacuum cleaner	Section 4.3

Figure 16 — Illustration of the examples used in this thesis.[2]

repeatedly in different contexts. Here, we introduce the examples, their implementation, parameters and — if existing — their variants.

The primary example for this thesis is a coffee machine and its sub-components. Coffee machines are widely used in households and offices and are a major contributor to productivity in industrial manufacturing. An additional example is a vacuum cleaner robot. The vacuum cleaner is a further example, which complements to the coffee machine to clean the kitchen as illustrated in Figure 16.

Figure 17 shows the coffee machine and its sub-components that form the running examples throughout this thesis. The behavioral function of the holistic coffee machine is represented as a Mealy machine in Section 3.7.1. The sub-components are the water tank, the boiler, and the power converter. The water tank is a system with continuous dynamics and is used to store water for the coffee machine and is presented in Section 3.7.2. The water tank can be extended to a two-tank system, which is introduced in Section 3.7.3. The last component is the boiler, which is used to heat the water in

[2]Image generated with ChatGPT

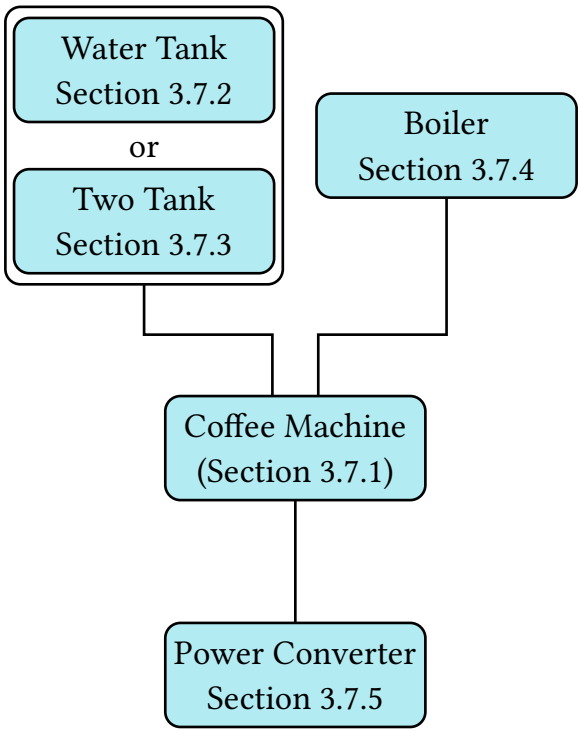


Figure 17 — Sub-components of the coffee machine. The sub-components are the water tank with an optional extension to a two-tank system, the boiler, and the power converter.

the tank to a defined temperature range as described in Section 3.7.4. While each of the sub-components is modeled as a separate system, they interact with each other to form the overall behavior of the coffee machine. This showcases that many CPSs are complex integrated systems, which do not stand alone, but serve specific functions within a variety of larger system. Water tanks, heating systems, and power converters are relevant sub-components in many other CPSs.

Note that physical units (such as meters, seconds, or volts) are used to describe system variables and signals for the examples, if applicable. However, in the experimental sections, units are omitted except for the axes labels in the figures. This is because the learning approaches operate on numerical data without explicit knowledge of the underlying physical units; all values are treated as unit-less numbers.

3.7.1 Coffee Machine

The coffee machine is an example of a system with a Mealy machine representation based on B. Steffen et al. [114]. This representation is shown in Figure 11. The system has the four input symbols BUTTON, CLEAN, POD, and WATER and the three output symbols ok, error, and coffee. The system has five states, where State 1 is the initial state and State 6 is an error state. A model of the coffee machine is implemented in LearnLib [115].

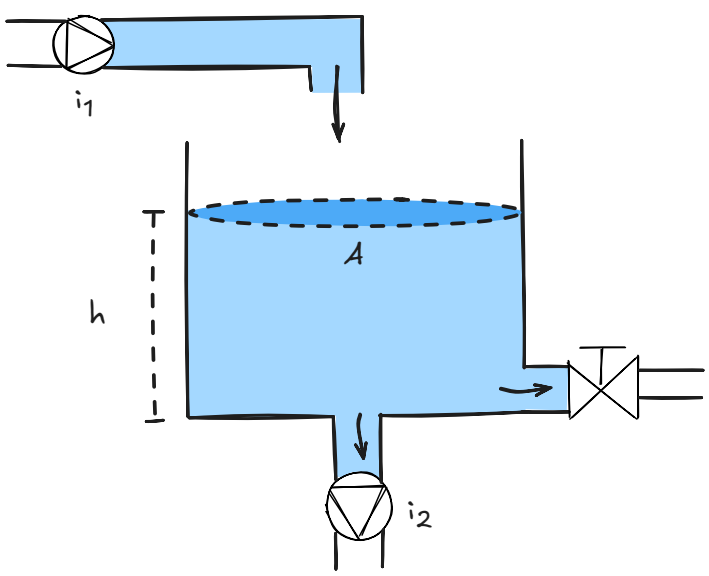


Figure 18 — Water tank system with inflow pump i_1, outflow pump i_2 for flushing operation and outflow valve for draining operation. The cross-sectional area of the tank is A and the height of the water in the tank is h.

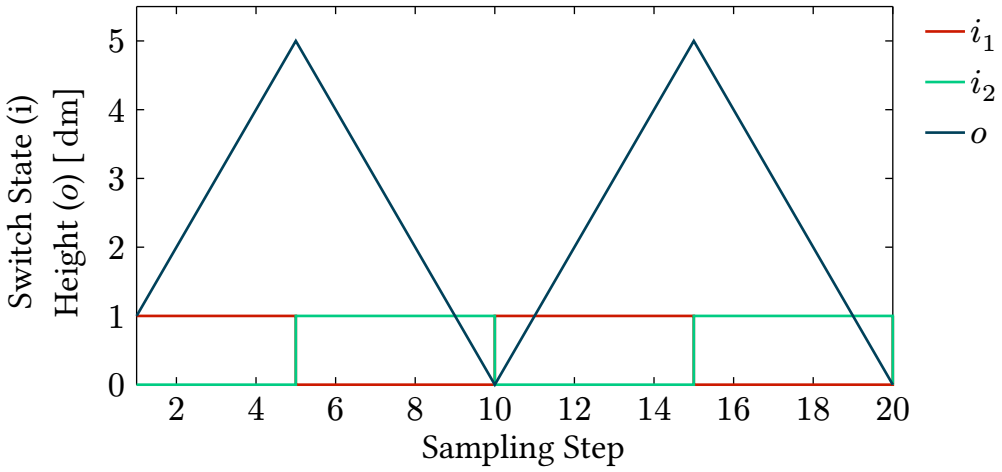


Figure 19 — Signals of the flushing operation of the water tank. The input signals are the states (1 or 0) of the valves at inflow i_1 and outflow i_2 pumps, while the output o is the water level of the reservoir in dm.

3.7.2 Water Tank

The water tank [31] is a system with continuous dynamics as shown in Figure 18. We consider two variants of the dynamics of the water tank for continuous and discrete modeling. The first variant is the continuous *flushing operation* of the water tank. The flushing operation uses two pumps, one for the inflow and one for the outflow. The input signals are the flow rates at inflow and outflow pumps while the output is the water level of the reservoir. The input and output flow rates are alternating, periodic rectangular functions describing modes of inflow and outflow as shown in Figure 19.

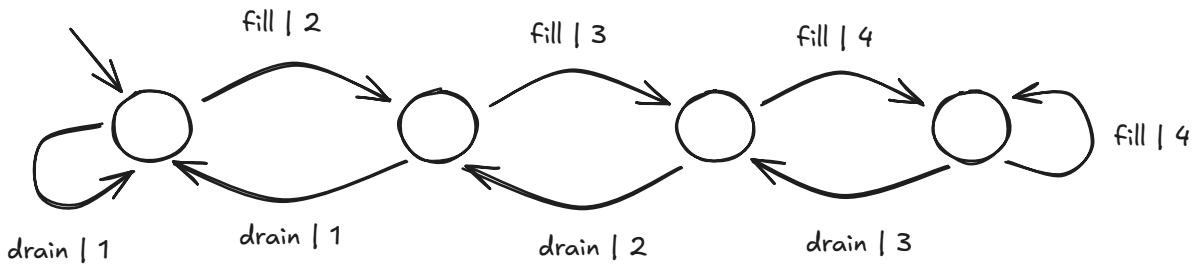


Figure 20 — Mealy machine representation of the water tank system with draining operation. The system has two input symbols **fill** and **drain** and the output symbols **1,2,3,4**, representing the water level of the tank.

Figure 21 — Two-tank system for water storage at the water supply and treatment center (Versorgungsbetriebe Helgoland) at Helgoland, Germany.

The system is modeled in Matlab and the total duration of one simulation run is 50 time units.

In addition to the continuous variant, we consider one *discretized variant* as shown in Figure 20. The discrete inputs are **fill** and **drain**, which represent the inflow and outflow mode of the tank, respectively — assuming a constant flow rate. The output symbols are **1**, **2**, **3**, **4**, which represent the water level of the tank — assuming that **4** is the maximum water level at which the tank overflows and **1** is the minimum water level including an empty tank.

3.7.3 Two Tank System

The two tank system [116] is a benchmark system for hybrid automata. Figure 21 shows a real-world two tank system for drinking water storage at Helgoland, Germany. The example is implemented in Matlab and consists of two tanks, with a pump pumping water into the first tank and a valve V_b that can be opened to let water flow from the

first to the second tank. The system has a controller for the height of the water in the first tank. The following differential equation describes the system:

$$\dot{h}_1 = \begin{cases} \frac{Q_p - C_{\text{vb}} \cdot \sqrt{h_1 - h_2}}{A}, & \text{if } h_1 > h_2 \text{ and } V_b \text{ open}, \\ \frac{Q_p + C_{\text{vb}} \cdot \sqrt{h_2 - h_1}}{A}, & \text{if } h_1 \leq h_2 \text{ and } V_b \text{ open}, \\ \frac{Q_p}{A}, & \text{if } V_b \text{ closed}, \end{cases} \tag{10}$$

where h_1 and h_2 are the heights of the water in the first and second tank, respectively, $\dot{h}_1$ is the derivative, i.e., the change in the water level, of the first tank. Q_p is the flow rate of the pump, C_{vb} is the flow coefficient of the valve, and A is the cross-sectional area of the first tank. The simulation involves noise and a realistic controller, which applies a zero-order hold to the observed height of the tank. The noise is additive noise with a maximum amplitude of 10^{-6} for all sensors.

3.7.4 Boiler

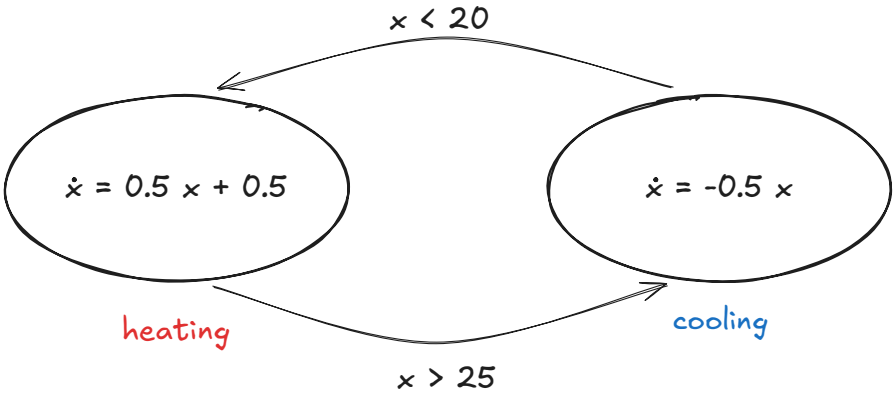


Figure 22 — Boiler system with controlled temperature range. The system has two modes: the heating mode and the cooling mode, which are described by differential equations.

The boiler is a system with continuous dynamics. The system consists of a water tank with a heating element and a temperature sensor. The heating element is controlled in a way that the water temperature in the tank is kept between a defined reference temperature or temperature range. We consider two variants of the system. The first variant is a *controlled temperature range* scenario. The temperature of the water in the tank is kept at a defined temperature range between 20 and 25 degrees Celsius. We model the system as a hybrid automaton with two modes: the heating mode and the cooling mode as shown in Figure 22, where x is the continuous water temperature.

The second variant is a *test scenario* where the reliability of the system is tested under varying reference temperatures. A temperature controller based on [117] is used. The input signal is the reference temperature, which changes every 50 time units to random values in a defined input domain. The output signal is the water temperature of the boiler. The simulation time of one run is 1400 time units.

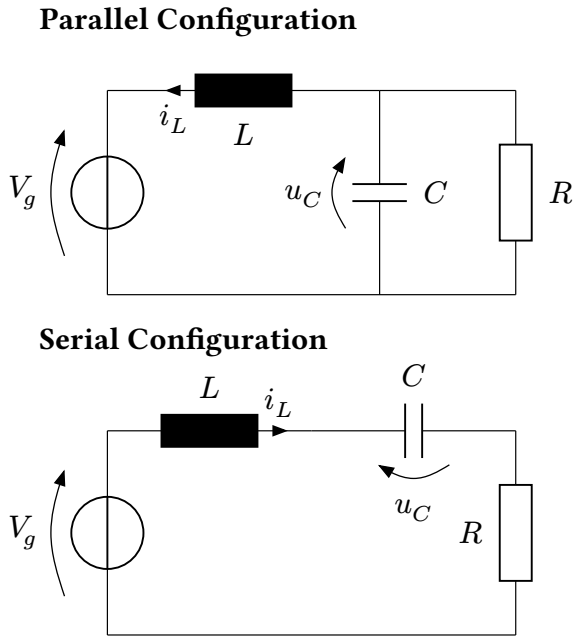


Figure 23 — Power converter consisting of a capacitor C, an inductance L, and a resistor R with parallel (top) and serial (bottom) configuration. The system has a controlled input voltage $v_s = \sigma V_g$ where V_g is a constant input and σ is a switching variable, which is either 1 or -1 [118].

3.7.5 Power Converter

The power converter [118], [119] is an electrical circuit with active components. We consider two variants. The first variant is a *generic power converter* with a controlled input voltage $v_s = \sigma V_g$ where V_g is a constant input and σ is a switching variable, which can be either 1 or -1. In addition to the voltage source, the circuit consists of a capacitor C, an inductance L, and a resistor R. The circuit can be either in parallel or in series configuration, as shown in Figure 23. As presented by N. Zaupa et al. [118], we use a transformed coordinate system to describe both configurations with the same equations. The following differential equation describes the system:

$$\dot{w} = \begin{pmatrix} 0 & \alpha \\ -\alpha & -\beta \end{pmatrix} w + \begin{pmatrix} 0 \\ \alpha \end{pmatrix} \sigma, \tag{11}$$

where $\alpha = \left(\sqrt{LC}\right)^{-1}$ and $\beta = (RC)^{-1}$ (parallel case) or $\beta = R \cdot L^{-1}$ (serial case). The transformed coordinates $w = [w_1, w_2]^T$ are constructed from the quantities in Figure 23 as

$$w_1 = \frac{v_c}{V_g} \quad \text{and} \quad w_2 = \frac{1}{V_g}\sqrt{\frac{L}{C}} \cdot i_c. \tag{12}$$

The inductance, capacity, and resistor have the values $R = 400\,\Omega$, $L = 8\,\mu\text{H}$, $C = 10.5\,\text{nF}$, and $V_g = 20\,\text{V}$.

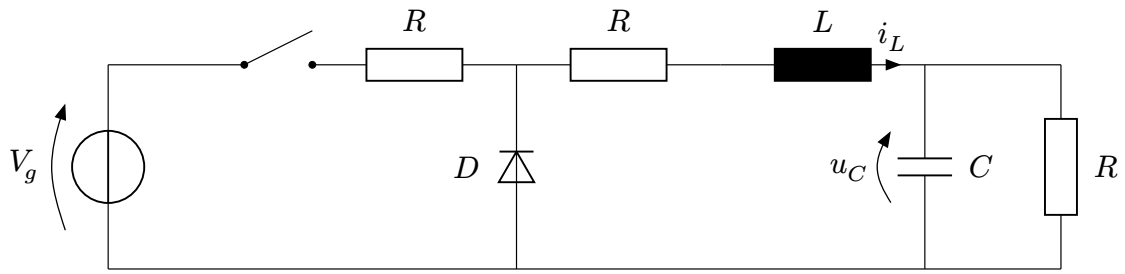


Figure 24 — Electrical circuit of a buck converter [119].

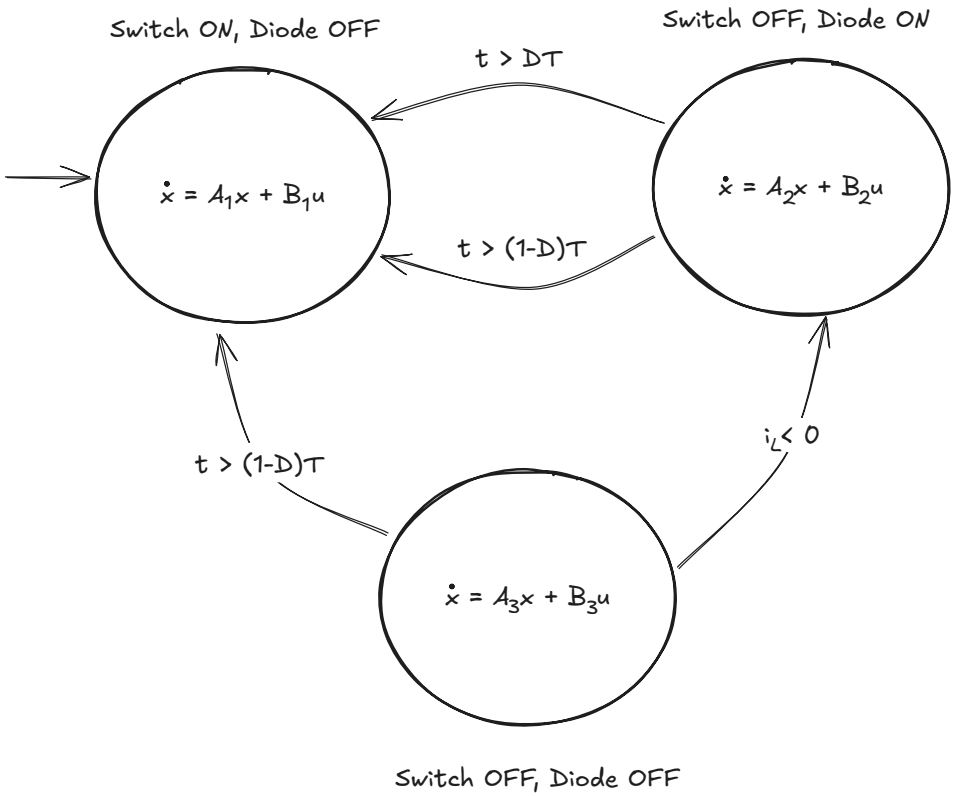


Figure 25 — Hybrid automaton representation of the buck converter [119]. The modes are governed by linear differential equations. The events for switching between the modes depends on the duty cycle D, the switching period T and the inductor current i_L. At every transition, the time t is reset to zero.

The second variant is a *buck converter* [119], [120] as shown in Figure 24 with $L = 2.65\,\text{mH}$, $C = 2.2\,\text{mF}$, and $R = 10\,\Omega$ [119]. The system has three modes depending on the switching state and the state of the diode, which are shown in Figure 25. The state vector is the inductor current i_L and the capacitor voltage v_C, i.e.,

$$x = \begin{bmatrix} i_L \\ v_C \end{bmatrix}. \tag{13}$$

We use the system and input matrices A_i, B_i with $i = 1, 2, 3$ including manufacturing tolerances as determined by O. A. Beg et al. [119].

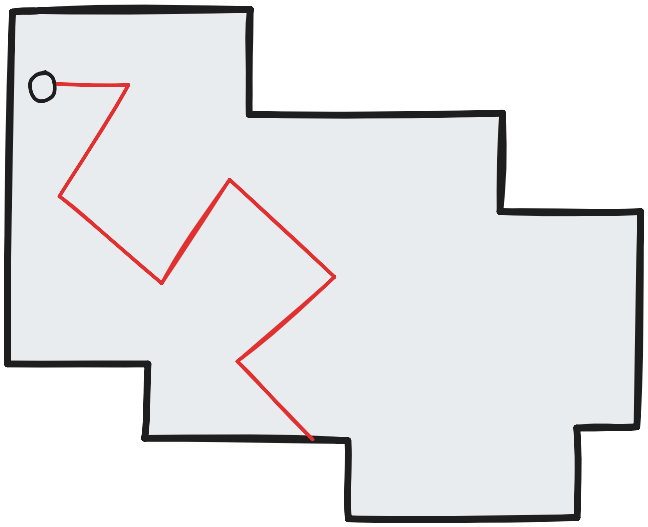


Figure 26 — Example trace of the vacuum cleaner robot. The robot moves inside a room (gray area) surrounded by walls (black lines). The red line shows the trajectory of the robot.

3.7.6 Vacuum Cleaner

We consider an indoor vacuum cleaner robot [121] as an example of a CPS. The simulation is implemented in Python. The robot acts in a random environment with walls. Figure 26 shows an example environment, where the red line gives the trajectory of the robot.

We have two different scenarios for the vacuum cleaner robot. The first variant is the *random autonomous cleaning* mode. The robot moves randomly in the environment until a fixed distance is covered. The robot always acts according to the same protocol — the robot moves straight until it touches a wall; upon touching a wall the robot drives back a small distance and rotates with a random angle. The robot then moves forward until it touches a wall again and repeats these actions until the maximum movement distance is reached.

Figure 27 shows a DFA model of this behavior. The alphabet of the automaton is $\Sigma = \{\texttt{move}, \texttt{touch}, \texttt{rotate}, \texttt{backwards}\}$. State 0 is the initial state and all states except for State 1 are accepting states.

In the second scenario, the *controlled cleaning* mode, the robot is controlled by a human operator. The input commands of the robot are 'move' and 'rotate' and define the alphabet $\Sigma = \{\texttt{move}, \texttt{rotate}\}$. The commands involve a distance and an angle to the robot, respectively. The controlled vacuum cleaner takes input traces of variable length. The input traces consist of movement and rotation commands, which the robot executes. The robot returns an output in the form of 'true' and 'false' depending on whether the given trace is valid or not. A trace is invalid if the robot at any point during the execution of the trace drives into a wall.

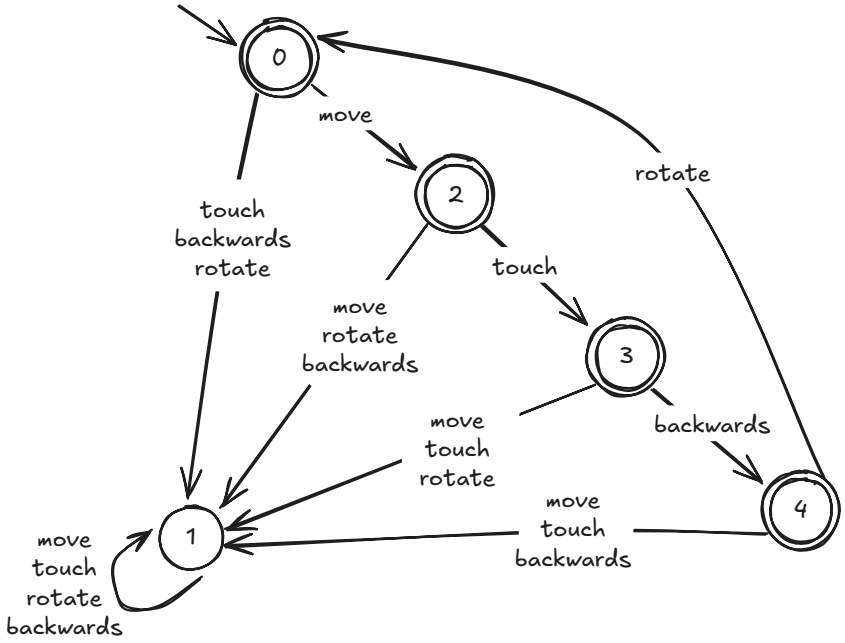


Figure 27 — Automaton model of the vacuum cleaner robot in random autonomous cleaning mode. The transitions between the states are labeled with the actions taken by the robot. The actions form the alphabet $\Sigma = \{\texttt{move}, \texttt{touch}, \texttt{rotate}, \texttt{backwards}\}$. State 0, State 2, State 3, and State 4 are accepting states representing the normal operation of the robot, while State 1 is non-accepting and represents all deviations from the normal operation.

Chapter 4

Identification of Deterministic Finite Automata

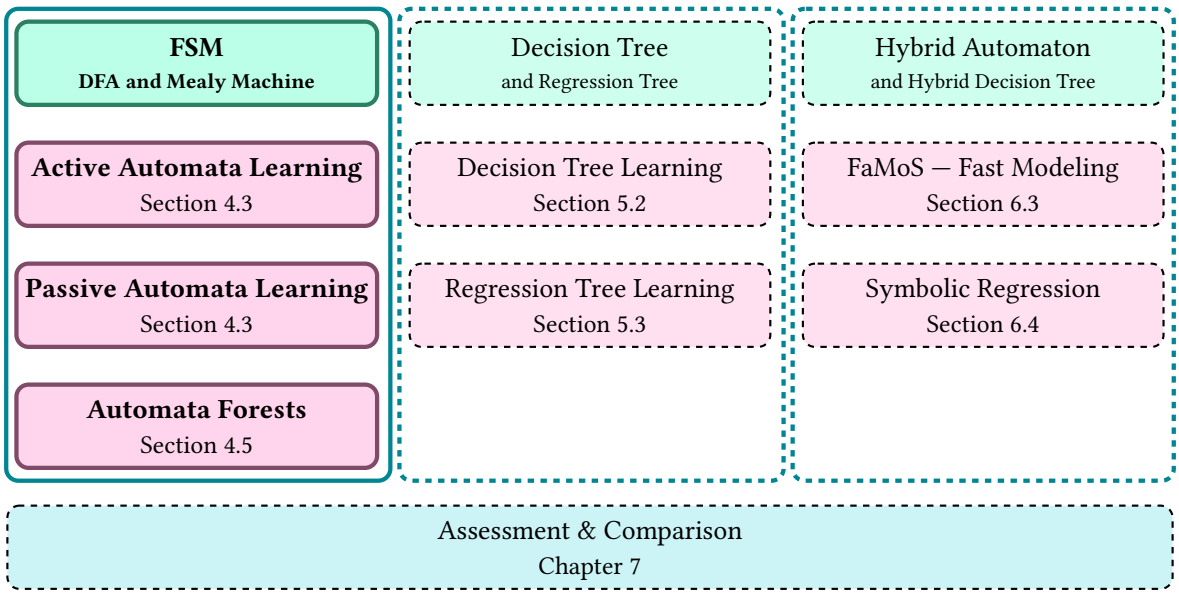


Figure 28 — Overview of the presented models (green) and learning approaches (pink) per model type in this work. Chapter 4 covers passive and active automata learning as well as automata forests.

The complexity of CPSs necessitates abstract models that capture essential behavioral patterns while remaining interpretable and compact. As established in our introduc-

tory LIDAR case study, the first modeling challenge is to determine whether classical discrete models can effectively represent CPS behavior when suitable abstractions are applied. In this chapter, we explore Deterministic Finite Automata (DFAs) as the foundation of the modeling hierarchy of this thesis — representing an interpretable and theoretically grounded approach to CPS modeling.

Figure 28 shows the scope of this chapter within this work. We focus on DFAs and Mealy machines, deterministic variants of FSMs and position automata learning as the starting point of our progression from discrete-deterministic through novel applications of decision trees to hybrid modeling approaches.

Automata models have proven successful in software and embedded systems, but their applicability to CPSs remains an open question. Our investigation bridges classical automata learning theory with the challenge of CPS modeling and answers our first research question **Q1 — In how far are classical learning methods for FSMs suitable for modeling CPSs?** by addressing the following interconnected questions:

- Q1.1 — How to abstract a CPSs such that it is learnable by automata learning?
- Q1.2 — Which chances and limitations of active and passive automata learning for CPSs exist?
- Q1.3 — How do the learned models support applications such as test case generation?
- Q1.4 — How to improve the applicability of automata learning on limited datasets and black-box systems?

We start with system abstraction — a critical prerequisite that determines whether the continuous-discrete interactions of CPSs are effectively captured in purely discrete models. The abstraction process must preserve behavioral information while enabling efficient learning, as demonstrated in our LIDAR example where environmental factors needed discrete categorization. For continuous components of CPSs, discretization represents one fundamental abstraction strategy. While abstraction and discretization constitute broad research domains beyond the focus of this work, we establish foundational concepts necessary for automata learning and other discrete modeling approaches explored in subsequent chapters. This connects to the decision tree approaches in the following chapter, where similar discretization challenges arise.

Subsequently, we analyze the capabilities and limitations of both passive and active automata learning paradigms when applied to CPSs based on L. Schammer et al. [27]. Our analysis reveals that both approaches are viable for CPS modeling, but require different abstraction strategies and present distinct practical trade-offs. These insights are illustrated through the vacuum cleaner robot example from Section 3.7.6, which demonstrates how the choice of learning paradigm influences the abstraction requirements and the resulting model characteristics.

For practical validation of learned automata models, we demonstrate the application for systematic testing of real-world systems [28]. Our application to an RTLS showcases how discrete models guide comprehensive system evaluation, while also

revealing the limitations that motivate the need for more expressive modeling approaches in subsequent chapters.

Finally, we address the robustness challenges inherent in learning from limited and potentially incomplete datasets. We present automata forests, which provide enhanced reliability for black-box modeling scenarios of CPSs. This is based on A. Krumnow et al. [29], which we extend by a detailed formalism and explicit learning characteristics.

4.1 Related Work

Starting with the seminal paper by Gold [66], grammatical inference algorithms are an important method to learn FSM models. E. M. Gold [66] guarantee the convergence to the target automaton when a learning set is enriched by more and more characteristic observations [122].

In Section 3.2, we introduce active and passive automata learning paradigms. There are several extensions and adaptations of both of these paradigms. Original automata learning algorithms are the L* algorithm for active automata learning and the RPNI algorithm for passive automata learning. The L* algorithm [67] uses a membership oracle and an equivalence oracle to learn a model of the system. The RPNI algorithm [123] is a passive learning algorithm, which learns a model from a set of observed traces. B. Steffen et al. [114] show how active automata learning can be used on reactive systems by learning the behavioral model of a coffee machine as a Mealy machine. The paper further discusses optimizations to learning automata more efficiently on reactive systems. M. Isberner et al. [124] introduce a new algorithm for active automata learning that achieves a better performance than the L* algorithm. In passive automata learning, there are several variants of the Elastic-Degenerate String Matching (EDSM) algorithm [125], e.g., S-EDSM [126] that improves the heuristics for state merges in the EDSM algorithm. P. García et al. [105] compare different state merging strategies for the RPNI algorithm. Strategies that depend on the learning data empirically show better performance than data-independent strategies.

Further, automata learning is applied in several domains, e.g., for cooperative CPS [127], for test case generation [9], or in combination with reinforcement learning [128]. K. Meinke [127] use active automata learning on a cooperative open cyber-physical system-of-systems (CO-CPS) to learn a model and apply model-based testing on the system. A. Maier [71] use an online passive learning algorithm to learn cyber-physical production systems. He further analyzes the additional requirements needed to learn a cyber-physical production system with a passive learning algorithm. These approaches learn the general behavior of a system. Additional applications are network modeling [128], analysis of safety barriers in reinforcement learning [129], and the identification of miss-classification in learned neural networks [130]. There exist approaches that apply automata learning for test case generation, e.g., for software [131], [132] or hybrid automata [9]. In B. Aichernig et al. [9] automata learning and neural networks

are combined to generate a model and test cases with the example of an autonomous driving scenario. P. García et al. [133] present automata teams, which learns multiple automata in parallel to achieve a more consistent accuracy of the result.

Further, combinations of automata learning with other techniques or paradigms exist. F. Howar et al. [10] introduce a method for automatic abstraction refinement. They propose a modified wrapper around the system that provides the needed abstractions and automatically refines the abstractions during the learning process. Other works similarly discuss the abstraction of complex, hybrid, or cyber-physical systems [9], [14], [134], [135]. When learning discrete models such as automata from CPSs, continuous signals of the system have to be discretized. In B. Aichernig et al. [9] the need of a mapping between concrete data and abstract symbols is discussed. J. L. Piovesan et al. [14] investigate whether an abstract system has a valid deterministic and discrete representation and show how the continuous dynamics of a system can be abstracted to a timed automaton. H. Zaatiti et al. [16] present multiple strategies for the abstraction of a hybrid system to discrete models, e.g., using a partitioning of the continuous state space or based on reachability constraints.

Here, we focus on the learning of discrete models of CPSs with automata learning. We compare the applicability and performance of active and passive automata learning on a concrete case study. An observation is that the two learning paradigms for active and passive learning require different abstractions of the system and have different chances and limitations.

4.2 System Abstraction

In this section, we discuss the necessity of abstraction for learning discrete models and provide an example of an abstraction through discretization of a continuous system. Throughout the thesis, we then assume that a suitable abstraction is given.

Abstraction is a crucial step when learning models of complex systems such as CPSs. Directly modeling the full detail of a CPS is often infeasible due to the complexity of the system and the presence of continuous dynamics. By abstracting the system to a discrete model — such as a DFA or Mealy machine — we focus on the aspects of behavior that are most relevant for the modeling task. This process involves deciding which parts of the system to represent and at what level of detail — balancing the need for a model that is both learnable by available algorithms and interpretable for analysis or application. In particular, abstraction enables us to isolate specific functionalities or subsystems, making it possible to construct models that are both manageable and meaningful, especially in the context of large or intricate systems.

We specifically focus on the abstraction for learning a Mealy machine model of continuous CPSs. This requires a discretization of the input and output signals of the system. The general principle of abstraction is shown in Figure 29. The solid rectangle in the upper row represents the black-box system, which receives inputs and returns outputs (hatched boxes). The system has an unknown inner state as well as an

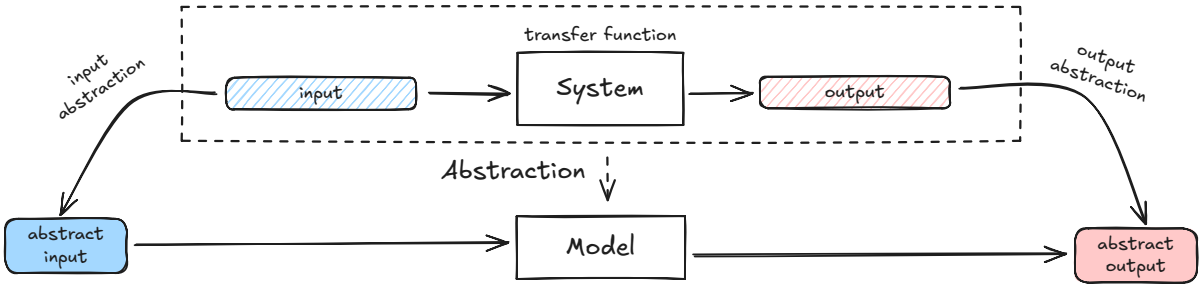


Figure 29 — An abstract representation of an SUL. The concrete inputs and outputs (hatched) are abstracted to abstract inputs and outputs (solid). This abstraction relates the concrete system and the abstract model.

unknown transfer function relating input and output. This transfer function represents the functionality of the system. The dashed rectangle is the concrete level of the system, which is abstracted to a model, the solid rectangle in the lower row. The model receives abstract inputs and returns abstract outputs (solid boxes).

For purely discrete models such as Mealy machines, continuous signals of (sub-)systems have to be discretized. Discretization leads to a loss of information and usually a trade-off between model complexity and granularity. S. Plambeck et al. [31] discusses the discretization of continuous signals for learning discrete models of continuous systems or hybrid automata. This involves a discretization of the time and value domains of the system. The abstraction is a value discretization on input and output signals, described by the functions

$$q_I : \mathbb{D}_I \to I, \quad q_O : \mathbb{D}_O \to O, \tag{14}$$

where I and O are finite sets of discrete, symbolic values and $\mathbb{D}_I \subseteq \mathbb{R}$ and $\mathbb{D}_O \subseteq \mathbb{R}$ are the continuous input and output domains of the system, respectively. Without loss of generality, the elements of I and O are one-dimensional.

We propose a discretization of the input and output domains by dividing them into sub-spaces $\mathbb{D}_I^1, \mathbb{D}_I^2, \ldots, \mathbb{D}_I^{|I|}$ and $\mathbb{D}_O^1, \mathbb{D}_O^2, \ldots, \mathbb{D}_O^{|O|}$ of $\mathbb{D}_I$ and $\mathbb{D}_O$, respectively. Each sub-space $\mathbb{D}_I^i$ and $\mathbb{D}_O^j$ then maps to a symbol $i_j \in I$ and $o_j \in O$, respectively. The discretization is a partition of the domains $\mathbb{D}_I$ or $\mathbb{D}_O$, i.e., the sub-spaces cover the whole domain and sub-spaces are pairwise disjoint. Using the sub-spaces, we define the discretization functions as follows

$$q_I(\boldsymbol{i}_c[t]) = i_j, \quad \text{for} \quad \boldsymbol{i}_c[t] \in \mathbb{D}_I^j, \quad q_O(\boldsymbol{o}_c[t]) = o_j, \quad \text{for} \quad \boldsymbol{o}_c[t] \in \mathbb{D}_O^j, \tag{15}$$

where t is a sampling time point and $\boldsymbol{i}_c[t] \in \mathbb{D}_I$ and $\boldsymbol{o}_c[t] \in \mathbb{D}_O$ are the continuous input and output signals, respectively.

Example 6: Assuming an output domain $\mathbb{D}_O = [0, 15] \times [0, 15]$, Figure 30 shows a regular discretization of the output space $\mathbb{D}_O$ into the sub-spaces $\mathbb{D}^1, \mathbb{D}^2, \ldots, \mathbb{D}^{11}$. Further, we define the discrete output symbols as $O = \{o_1, \ldots, o_{11}\}$. The discretization function is defined as follows

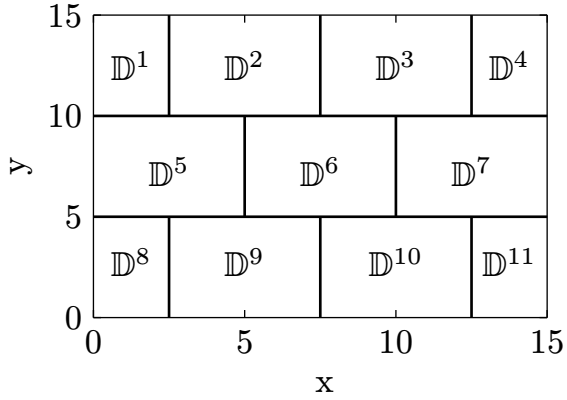


Figure 30 — Example of sub-spaces for the two dimensions x and y with categorization spaces of width and height 5.

$$q_O(\langle x, y \rangle) = \{o_j \quad \text{if } \langle x, y \rangle \in \mathbb{D}^j. \tag{16}$$

We define a neighborhood in $\mathbb{D}_O$ to determine the distance between the discrete output symbols. This neighborhood is the set N_O^i of neighbor symbols for the output o_i. Based on neighborhood sets, we define a distance between output symbols:

$$\text{dist}(o_i, o_j) = \begin{cases} 0, & \text{if } o_i \equiv o_j, \\ 1, & \text{if } o_j \in N_O^i, \\ 1 + \min\limits_{o_k \in N_O^i} \text{dist}(o_k, o_j), & \text{else.} \end{cases} \tag{17}$$

Example 7: Continuing the above example the intuitive neighborhoods for the categorization sub-spaces of Figure 30 are

$$\begin{aligned} &\mathcal{N}_O^1 = \{o_2, o_5\}, \mathcal{N}_O^2 = \{o_1, o_3, o_5, o_6\}, \mathcal{N}_O^3 = \{o_2, o_4, o_6, o_7\}, \\ &\mathcal{N}_O^4 = \{o_3, o_6, o_7\}, \mathcal{N}_O^5 = \{o_1, o_2, o_6, o_8, o_9\}, \ldots \end{aligned} \tag{18}$$

For time discretization, we apply a fixed sampling period t_S. The time and value discretization of the system has to be chosen such that the relevant information of the system is preserved. In some cases, sampling and discretization lead to non-determinism and ambiguous observations. This is illustrated in Figure 31. Instead of the trace 1,2,3,4,3,..., the trace 1,2,3,3,3,... is observed in the second repetition of the triangular signal. This happens because the sampling period t_S is too large to capture the change from 3 to 4. Thus, the output observed after the sequence 1,2,3 is ambiguous and can be either 3 or 4.

To apply the RPNI algorithm (passive learning) or the L*-algorithm (active learning) for learning FSMs, it is required to obtain unambiguous observations from the system. In the remainder of this chapter, we assume that the system has been suitably abstracted so that these learning algorithms can successfully construct an FSM model.

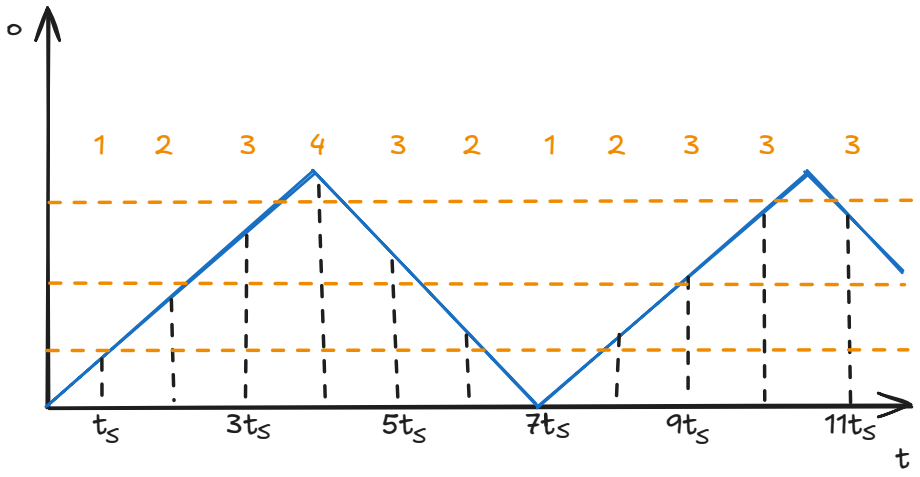


Figure 31 — Example for the discretization of a continuous output trace o with a sampling period t_S. The continuous values are discretized to four output symbols $o_1 = 1$, $o_2 = 2$, $o_3 = 3$, and $o_4 = 4$, indicated in orange above the trace.

4.3 Capabilities of Passive & Active Automata Learning for Cyber-Physical Systems

In L. Schammer et al. [27], we analyze how active and passive automata learning applies to CPSs and how the learned automata are interpreted in the context of the learning problem. We first, specify the learning scenario for active and passive automata learning. Then, we discuss the example of the vacuum cleaner in more detail. We observe that the practical learning scenarios differ in their approach to system interaction and trace collection as well as the abstraction level of the system.

4.3.1 System Observation & Learning Strategy

We use the RPNI algorithm [123] for passive automata learning. Passive automata learning requires a set of pre-collected traces of the SUL. We observe traces from the system as introduced in Section 3.4. A trace consists of an input and an output trace, i.e.,

$$\boldsymbol{s} = \begin{bmatrix} \boldsymbol{i} \\ \boldsymbol{o} \end{bmatrix}, \tag{19}$$

where

$$\boldsymbol{i}[j] \in I, \boldsymbol{o}[j] \in O \quad \forall j \in \{1, ..., |\boldsymbol{s}|\}. \tag{20}$$

The number of samples per trace is not fixed, i.e., the length of the trace is flexible. The input and output alphabets, I and O are finite, categorical sets. In the case where some inputs or outputs are continuous, the sets I and O represent the already abstracted, i.e., discretized values.

The learning set for passive automata learning is a set $\mathcal{O}$ of traces. Additionally, the input alphabet Σ is an input to the learning algorithm. This alphabet Σ is a finite set

of symbols, which are used to represent the input signals of the system. The alphabet is defined by the abstraction of the system as discussed in Section 4.2.

Active automata learning requires a direct interaction with the SUL, i.e., traces are generated by querying the system. This is often challenging for CPSs, because accessibility and controllability of the system has to be ensured. Thus, only the accessible and controllable parts of a CPS can be learned. Further, depending on the SUL, the execution time for a single query can be very long. In addition to online queries, active automata learning algorithms require an equivalence oracle, which checks whether the learned model is equivalent to the SUL. However, in practice such an equivalence oracle usually does not exist for complex and black-box systems. We use an approximation of an equivalence oracle, which checks the equivalence of model and SUL on a set of random traces. This random equivalence oracle has two parameters for the length and number of random traces that are generated for the equivalence check. Both parameters trade-off between learning time and completeness of the equivalence check. We use the active automata learning algorithm L* [67] for a case study on the vacuum cleaner example in Section 3.7.6.

4.3.2 Case Study: Vacuum Cleaner

Setup
Examples:
- Vacuum cleaner from Section 3.7.6

Language: Java
Scenarios:
- Random autonomous cleaning of the vacuum cleaner (active and passive learning)
- Controlled cleaning of the vacuum cleaner with fixed movement distance and rotation angle (active learning)

Learning Set:
- Passive learning: two traces with 5 m of distance covered by each run
- Active learning: variable number of learning samples through active queries to the vacuum cleaner simulation

Device: IntelCore i7@2.90GHz, 32GB RAM

We evaluate the learning algorithms on the simulated vacuum cleaner robot as introduced in Section 3.7.6 and compare the learning scenarios of active and passive learning as a representative case study for learning discrete models of CPSs. The vacuum cleaner robot is a simple CPS with a discrete behavior. Thus, challenges and limitations of the learning algorithms are clearly visible and likely to occur in more complex CPSs as well.

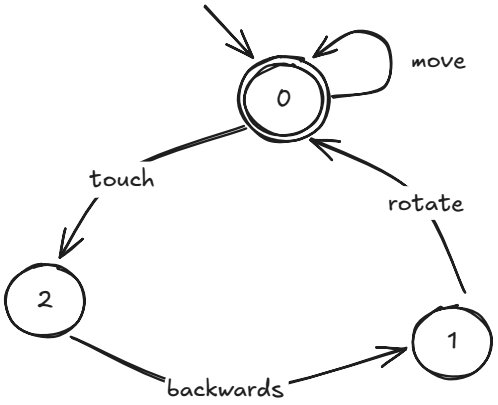


Figure 32 — Learned automaton model of the vacuum cleaner robot in random autonomous cleaning mode. The input alphabet is $\Sigma = \{\texttt{move}, \texttt{touch}, \texttt{rotate}, \texttt{backwards}\}$ and State 0 is the accepting state.

The goal is to learn the sequential behavior of the vacuum cleaner robot. The passive learning algorithm uses the robot in a random autonomous cleaning scenario, where the robot automatically cleans the room without any external control inputs. The input alphabet for passive learning is $\Sigma = \{\texttt{move}, \texttt{touch}, \texttt{rotate}, \texttt{backwards}\}$. This abstraction of the sensors and actuators captures the essential actions of the robot in the cleaning process.

Figure 32 shows the learned automaton. The automaton represents the behavior of the original model from Section 3.7.6 and Figure 27 only partially. The robot moves until it touches an obstacle, drives a small distance backwards and rotates before moving forward again, which is equivalent to a path through the original model in Figure 27. Nevertheless, only State 0 is an accepting state, while the other states should be accepting as well, because the robot could stop at any state in the process. The reason for this is that all traces in the learning set end with a move command. Further, the DFA misses the non-accepting states for the traces that are not possible. The observation of negative traces, i.e., traces that are not executed by the system, is not included in the observation of the nominal behavior and, thus, not learnable by the passive learning algorithm.

We learn the same behavior of the vacuum cleaner robot with an active learning algorithm in form of a DFA. The learned automaton is shown in Figure 33. The learned automaton is equivalent to the original behavior of the vacuum cleaner robot in Section 3.7.6 and Figure 27. This is because the equivalence oracle checks the equivalence of the learned automaton with the system. If the hypothesis automaton is not equivalent to the system, the equivalence oracle returns a counterexample trace. This counterexample is executed as a query to the system and the learner refines the hypothesis automaton accordingly. The robot is used in an autonomous cleaning scenario, where the executed commands are not controlled by the learner. Instead, the

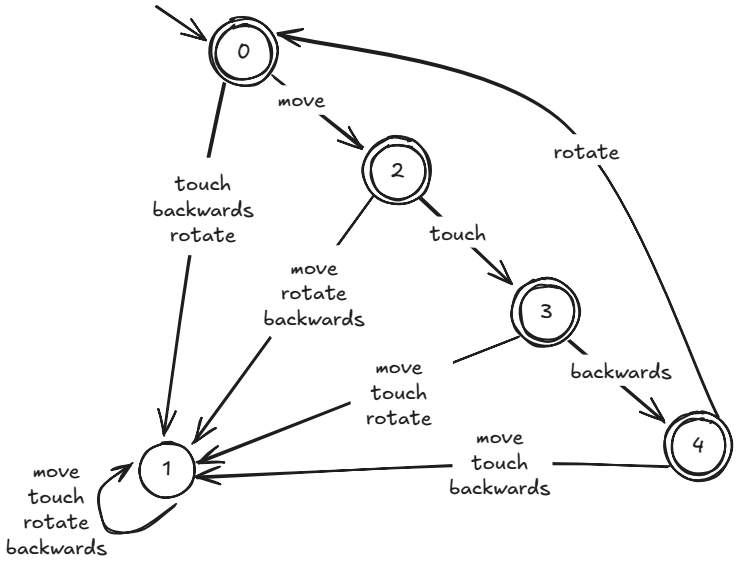


Figure 33 — Learned automaton model of the vacuum cleaner robot in controlled cleaning mode and input alphabet $\Sigma = \{\texttt{move}, \texttt{touch}, \texttt{rotate}, \texttt{backwards}\}$ and States 0, 2, 3, and 4 are accepting states.

learner queries the system whether a given trace of commands would be executed by the robot in autonomous mode.

Another active learning scenario for the robot is in a controlled cleaning scenario. Here, the queries of the active learner execute commands on the vacuum cleaner robot to explore the behavior. The following abstraction of real-world movements and sensors to the observed symbols from the alphabet of the automaton is applied: for every rotate operation, the rotation angle is fixed to 180°. The movement distance is set to 1 m, 0.5 m and 0.25 m over three different learning scenarios. A rotation angle of 180° indicates that the robot drives back and forth on a single line. The area, the robot can reach is limited by the movement distance and rotation angle in addition to the layout of the generated environment. From the start position, the robot has the possibility to drive a distance of 1 m in either direction.

Figure 34, Figure 35, and Figure 36 show the learned automata for the different movement distances. The states of the learned automata encode the coordinates of a position in the environment in relation to the starting position of the robot and the current orientation. Figure 34 shows the learned automaton for a movement distance of 1 m, where the robot can reach positions from −1 m to 1 m on the x-axis. In Figure 35, State 1 encodes a position of 0.5 m with orientation in positive x-direction and State 4 a position of 0.5 m with orientation in negative x-direction. The same interpretation applies to the automata in Figure 36, which has eight states encoding positions and orientation of the robot. Comparing the three automata reveals that, as the movement distance decreases, the number of states increases to capture the finer-grained positions and orientations.

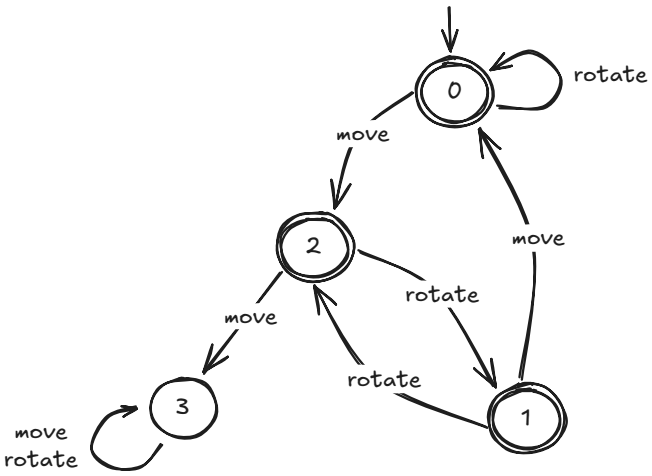


Figure 34 — Learned automaton model of the vacuum cleaner robot in controlled cleaning mode. The input alphabet is $\Sigma = \{\texttt{move}, \texttt{rotate}\}$ and States 0, 1, 2 are accepting states. The move command has a distance of 1 m and the rotate command has a rotation angle of 180°.

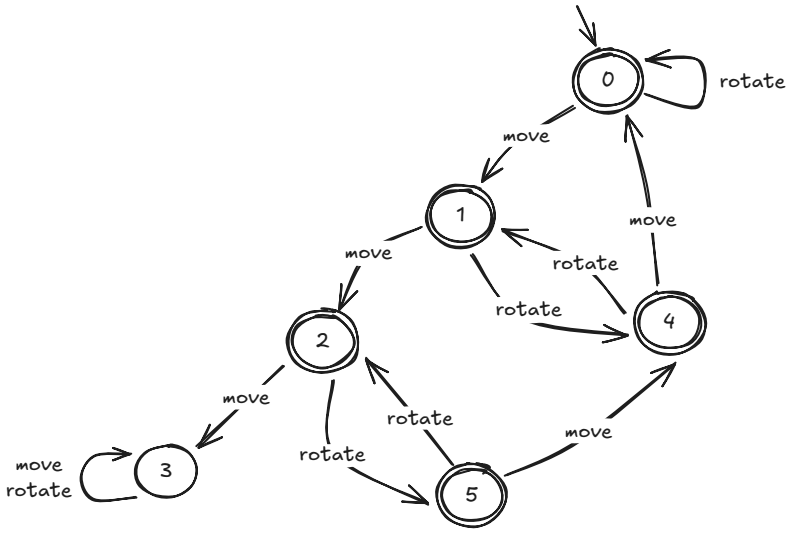


Figure 35 — Learned automaton model of the vacuum cleaner robot in controlled cleaning mode. The input alphabet is $\Sigma = \{\texttt{move}, \texttt{rotate}\}$ and States 0, 1, 2, 4, 5 are accepting states. The move command has a distance of 0.5 m and the rotate command has a rotation angle of 180°.

Table VI shows execution times for learning a model for the different movement distances. The time needed to learn a model in Table VI indicates that already for a simple abstraction of a simulated vacuum cleaner the learning process takes a long time, because a lot of queries are simulated on the system.

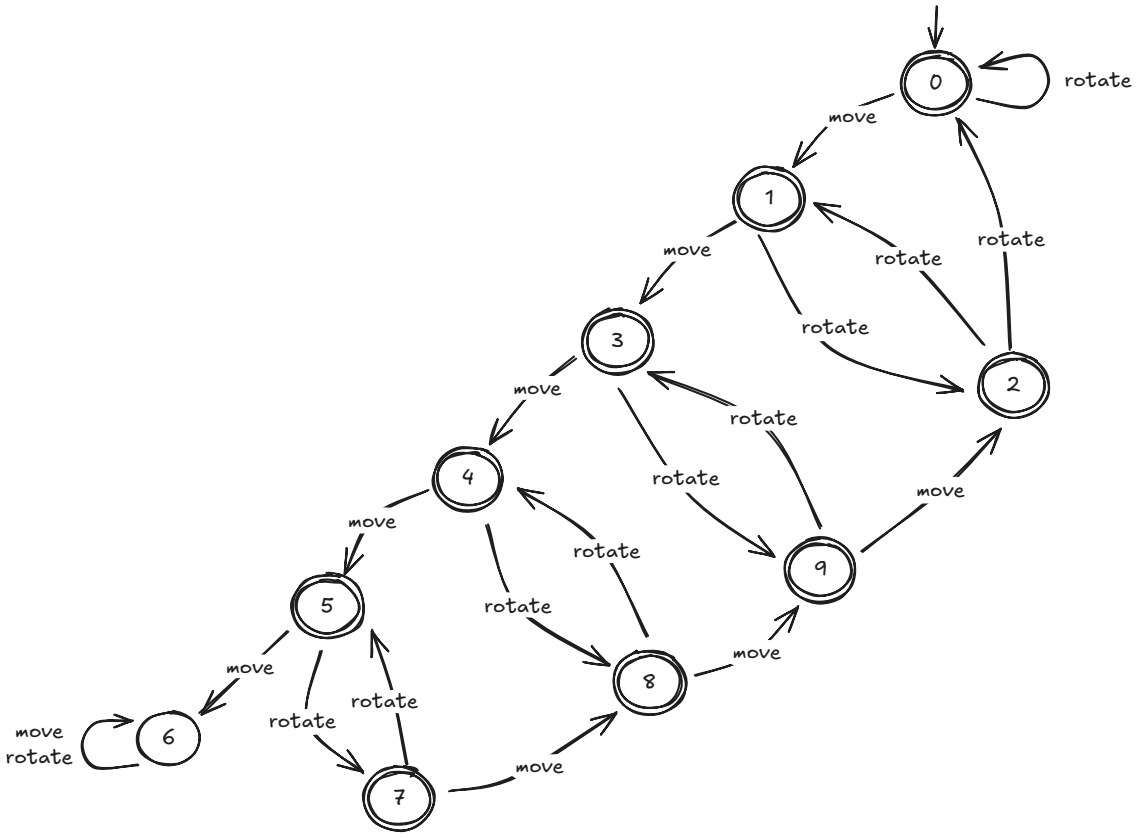


Figure 36 — Learned automaton model of the vacuum cleaner robot in controlled cleaning mode. The input alphabet is $\Sigma = \{\texttt{move}, \texttt{rotate}\}$ and all states except for State 6 are accepting states. The move command has a distance of 0.25 m and the rotate command has a rotation angle of 180°.

Table VI — Learning results with 100 traces of length 10. The distance covered by a `move` command, the time needed to learn the automaton and the number of queries needed to learn the automaton are shown.

Distance [m]	Time [s]	Number of Queries
1.0	75	213
0.5	113	213
0.25	227	648

4.3.3 Summary & Discussion

The case study illustrates that both active and passive automata learning algorithms are suitable for constructing discrete models. Nevertheless, our case study identifies limitations of the application of the learning paradigms to CPSs. The abstraction and observation of the system depends on the learning paradigm. In practice, the implementation of active automata learning on a CPS is often challenging, as the system

has to be controllable and accessible. The accessibility and controllability of the system influences the learning process. Already for the small case study of the vacuum cleaner robot, the learning process is time-consuming and requires several hundred queries in the active learning scenario, despite the small number of states and transitions in the learned automata. Again, the abstraction as well as the implementation of the equivalence and membership oracles have to be chosen carefully to ensure that the size of the automata is manageable and the required amount of learning data does not explode.

4.4 Test Case Generation with Automata Learning

In S. Plambeck et al. [28], we present a method for generating test cases for a given system using automata learning. We rely on established algorithms for coverage on FSMs such as state coverage and transition coverage [136]. The output of the automated test generation are traces of input features.

State coverage (SC): State coverage returns a set of input traces $\mathcal{O}_{SC}$ that ensure that all states of the FSM are visited at least once, when applied to the system.

Transition coverage (TC): Transition coverage returns a set of input traces $\mathcal{O}_{TC}$ that ensure that all transitions of the FSM are passed at least once, when applied to the system.

State coverage and transition coverage enable the automatic generation of test cases using only the learned automaton, without requiring additional system knowledge. State coverage typically results in fewer test cases than transition coverage as it only requires visiting each state once. Transition coverage explicitly aims to traverse every transition in the automaton, providing a more thorough exploration of the system behavior. There may be multiple possible sets of input paths that achieve full transition coverage, depending on the structure of the automaton. Selecting among these sets can be guided by additional criteria, such as minimizing the total number of test cases or prioritizing critical transitions.

For both approaches, the result of the test case generation is a set of traces of inputs. These input features represent abstract, potentially discretized inputs to the system. Thus, they do not directly represent a test case that is executable on the system under test and a mapping between input traces and executable test cases is needed.

We apply the test case generation to RTLSs for indoor localization. RTLSs are measuring systems that have a strong interaction with physical effects of their environments like light conditions, temperature, surrounding structures, vibration, infrastructure. Localization systems are often proprietary and their inner working is hidden from the user. Usually, an extensive testing of localization systems is needed to estimate the localization quality in a given environment. Our goal is to derive test cases for the system to systematically determine the localization quality in a given

environment. For this systematic testing, we need a model for the relation between environmental influences and the localization quality, i.e., the accuracy of the location estimation. Our aim is to build interpretable models representing environmental influences which enables automated test case generation.

We present a work-flow for automated test case generation for black-box systems and discuss relevant aspects of this procedure within the case study of RTLSs. The procedure is as follows.

1) Automata learning is used to build an abstract model of the given system.
2) We apply coverage-guided algorithms to derive relevant paths on the learned automaton model.
3) We convert the paths to executable test cases for the system under test.

The procedure of model building and automated test case generation builds on established algorithms for automata learning as presented in Section 3.2 and coverage-guided test case generation. We contribute a new method that joins automata learning and established coverage algorithms for automated test generation.

4.4.1 Experimental Setup & Data Aggregation

The case study is conducted with an RTLS based on a LiDAR sensor. The measurement is based on the *time of flight* method. The distances from the sensor to the contours of the environment are determined based on the time between emission and detection of a reflected laser impulse [137]. The measurement takes place in several scan layers with a field of view of 180°. Comparing the measurements to a prerecorded map gives an estimated position and orientation of the sensor and, thus, the entity under localization. Influences on the quality of the localization are the spatial environment and its dynamics, the map quality, the lighting conditions, or the movement of the sensor. Since the given system is proprietary with highly complex inner workings and unknown error propagation, it is considered a black box.

The experiment is conducted at the testing facility of the Institute for Logistics Engineering at Hamburg University of Technology on a rectangular area of $80\,\mathrm{m}^2$. The layout of the testbed is shown in Figure 37. To gain the ground truth position of the entity under localization, an optical motion capture system is installed [138]. The difference in the localization of the RTLS and this reference system is the localization error of the RTLS. The localization sensor is attached to a trolley, which is manually pushed along a predefined trajectory, i.e., a path on the rectangular testbed. The position information is received with a frequency of 20 Hz. Time synchronization between the systems is achieved by using the precision time protocol.

One run of the experiment results in a set of 5,000 to 15,000 samples, depending on the length of the trajectory of the run. Each sample contains a pair $\langle \boldsymbol{x}_{\mathrm{RTLS}}, \boldsymbol{x}_{\mathrm{gt}} \rangle$ of absolute position and orientation measured by the localization and the reference system. From this data, we derive the absolute position error $\boldsymbol{e}_{\mathrm{pos}} = \left| \boldsymbol{x}_{\mathrm{RTLS}} - \boldsymbol{x}_{\mathrm{gt}} \right|$ of the localization system.

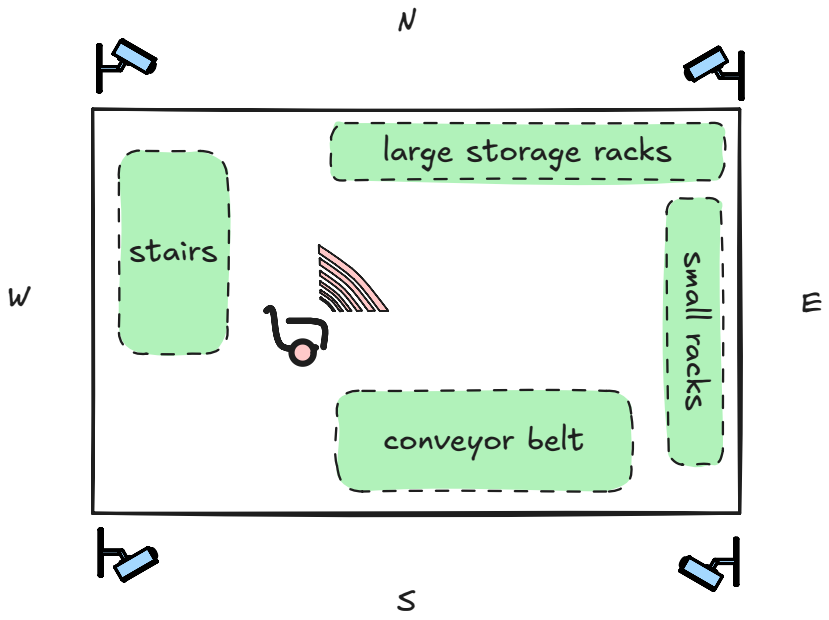


Figure 37 — Layout of the experimental setup for the case study at the Institute for Logistics Engineering at Hamburg University of Technology. The reference localization system is indicated by the cameras in the corners of the testbed. The RTLS is installed on a trolley, which is manually pushed along a predefined trajectory.

4.4.2 Automata Learning

Setup
Examples:
- Real-Time Localization System (RTLS) with LiDAR sensor

Language: Java
Learning Set: 5 traces of 10,000 to 15,000 raw samples abstracted to approx. 100 observations
Test Set: 3 traces
Metrics:
- Model learning: Localization error, i.e., the mean absolute deviation of the localization system from the ground truth position discretized in two classes: non-critical errors and critical errors.
- Test case generation: State coverage and transition coverage on the learned automaton.
- Test case execution: Number of generated test cases, i.e., the number of traces that are generated by the coverage algorithms.

Our learning procedure starts from the learning set $\mathcal{O}$ of traces resulting from the data acquisition runs of the RTLS. For automata learning a suitable abstraction of the data

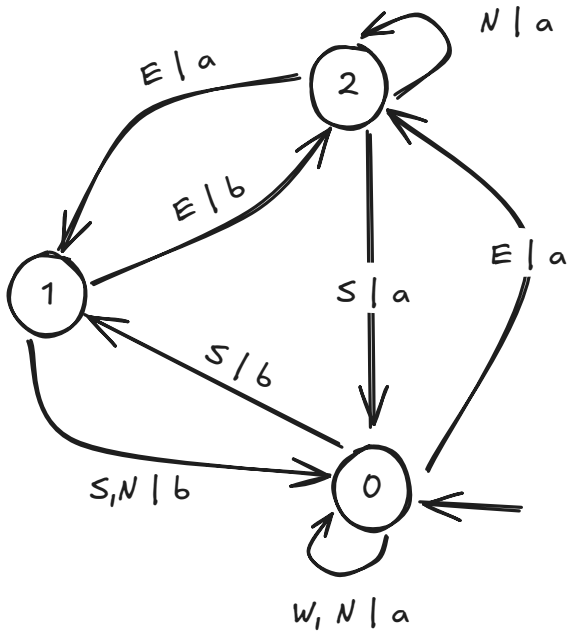


Figure 38 — Learned Mealy machine for the localization system. The automaton has three states $Q = \{0, 1, 2\}$, four input symbols $I = \{N, E, S, W\}$ and two output symbols $O = \{a, b\}$.

is needed as discussed in Section 4.2. For this case study, we use the following steps to prepare and abstract the data for automata learning:

1) feature selection,
2) discretization, and
3) grouping of consecutive duplicates.

Step 1) defines the information used for learning. Specifically, we select the inputs and outputs of the model. Here, we use the localization error as the output to describe the localization quality [139]. The orientation of the system under localization is the input feature and describes which direction the system is oriented in the horizontal plane. Both of these features are real-valued signals and, thus, have to be discretized in Step 2). The localization error is mapped to two discrete classes, which we call a and b, where a represents non-critical errors $e \leq 0.15\,\mathrm{m}$ and b represents critical errors $e > 0.15\,\mathrm{m}$. The criticality depends on the use-case and is defined by the user. Here, we assure a safe pickup of trays. The orientation is mapped to 4 sectors (north N, east E, south S, west W). Step 3) merges consecutive equivalent values to reduce observed traces to relevant discrete transitions.

After data acquisition and preprocessing, we use passive automata learning to generate an abstract Mealy machine model. Figure 38 shows the automaton that is learned from the data of the RTLS. A critical error b is observed for the transitions from State 0 to State 1 and from State 1 to State 2. All other transitions correspond to non-critical errors a.

Table VII — Number of generated test cases for state coverage $\mathcal{O}_{SC}$ and transition coverage $\mathcal{O}_{TC}$ for different input abstractions with different numbers of input symbols $|I|$ and output symbols $|O|$.

$\|I\|$	$\|O\|$	$\|\mathcal{O}_{SC}\|$	$\|\mathcal{O}_{TC}\|$
4	2	6	25
6	2	8	43
8	2	8	54

4.4.3 Test Case Generation on the Automaton Model

For the automaton from Figure 38, we obtain the input traces

$$\mathcal{O}_{SC} = \{\langle N \rangle, \langle S \rangle, \langle S, E \rangle\} \tag{21}$$

for state coverage. Transition coverage results in the set

$$\mathcal{O}_{TC} = \{\langle N \rangle, \langle E, N \rangle, \langle E, E \rangle, \langle E, S \rangle, \langle S, S \rangle, \langle S, E \rangle\} \tag{22}$$

of input traces for testing. Each element of the sets $\mathcal{O}_{SC}$ and $\mathcal{O}_{TC}$ provides a test case on the abstract representation, which are traces of orientations in the horizontal plane. For example, the trace $\langle S, E \rangle$ corresponds to a test case, where the system is first oriented towards the south and then towards the east.

For the concretization of the abstract test to executable tests on the RTLS, we map each orientation to a movement of a unit distance in the direction of this orientation. This maps, for example, $\langle S, E \rangle$ to the test case '*Move one Meter South, Move one Meter East*'. With this, the test cases generated for the abstract model are transformed into concrete test cases, which can be applied to the system under test.

Testing with the generated test cases is done by executing the test in a reset, guided testing mode, where the system is reset to a known state before each test case is executed guided by the input trace. The test cases serve to test the system under test in a way that all states and transitions of the learned automaton are covered. This supports the identification of critical input-output combinations of the system and is especially useful when implementing a system in a new environment.

The level of abstraction influences the resolution of the model and, thus, the number of generated test cases. In this example, we use a low level of abstraction with only four input symbols and two output symbols. Table VII shows the number of generated test cases for an increased resolution in the input abstraction, i.e., with more input symbols.

The result of this case study showcases how automata learning enables coverage-guided test case generation for CPSs. The learned automaton model provides a structured representation of the system's behavior, enabling the generation of relevant test cases. The generated test cases ensure comprehensive coverage of the states and transitions of the system, facilitating systematic testing. A crucial step is the

abstraction of the system behavior, which influences the resolution of the model and the number of generated test cases as well as the concretization of the abstract test cases back to executable tests. Through behavioral coverage, the test case generation is guided by the model and the system behavior and identifies relevant testing modes.

4.5 Automata Forests

In the following, we focus on passive automata learning. Passive learning algorithms learn a model from a set of observed traces. Thus, limitations for active learning, such as accessibility and controllability of the system, do not apply. Passive automata learning benefits from the strong theoretical foundation of grammatical inference such as convergence guarantees to the correct model. Nevertheless, these guarantees only hold under the assumption that a suitable learning set is used, i.e., the learning set covers the language of the system. When learning an automaton for CPSs, the learning scenario often considers the system as a black-box system. Thus, the learning set is not necessarily a complete representation of the system's behavior. Especially, the information on the quality and amount of information in the learning set is not known.

A. Krumnow et al. [29] shows how subsets of the learning set can be used to learn a set of automata, which we call an *automata forest*. As automata forests learn multiple automata, they explore the solution space more thoroughly than traditional automata learning algorithms. The goal of automata forests is, thus, to identify and select better approximations of the target automaton.

4.5.1 Formalization & Learning Algorithms

An automata forest $\mathcal{F} = \{\mathcal{A}^1, ..., \mathcal{A}^m\}$ consists of multiple automata, i.e., *instances* $\mathcal{A}^i$ for $i \in [1, ..., m]$ and an inference function h. Every instance is an automaton, learned from a random subset $\mathcal{O}_j$ of the original learning set $\mathcal{O}$. The inference function h determines the output of the forest for a given input trace. We present two different inference algorithms, which we call *Performance Metrics*-based approach and *Majority Vote*-based approach.

The first function h_{PM} determines a single automaton $\mathcal{A}_{\mathrm{PM}} \in \mathcal{F}$ in the forest that is the most promising candidate among the instances and returns the output of this automaton, i.e.,

$$h_{\mathrm{PM}}(\boldsymbol{i}) = \mathcal{A}_{\mathrm{PM}}(\boldsymbol{i}). \tag{23}$$

We call this approach the *Performance Metrics*-based approach as we use performance metrics to determine the most promising automaton. The performance metric is represented by a function $g : \mathbb{A} \times (\Sigma^*)^P \to \mathbb{R}$, which maps an automaton and a set of traces to a real number. The function g evaluates how well an automaton predicts on a given validation set. We require that g returns an optimal performance score only if the automaton predicts all traces in the validation set correctly. Here, we use metrics determining the maximum number of correctly predicted traces (maximization metric)

```
1: function buildAutomataForest(𝒪, m, ρ)
2:     for i ∈ [1, ..., m] do
3:         𝒪_i ← createSubset(𝒪, ρ)
4:         𝒜_i ← learnAutomaton(𝒪_i)
5:     end
6:     return 𝒜_1, ..., 𝒜_m
7: end
```

Listing 2 — Generic algorithm for building an automata forest. The steps are the creation of subsets and the learning of automata from these subsets.

or the Hamming distance of a predicted and true output trace (minimization metric). Based on the performance metric, the candidate automaton A_{PM} is determined as follows

$$\mathcal{A}_{\mathrm{PM}} = \begin{cases} \underset{\mathcal{A}_j \in \mathcal{F}}{\operatorname{argmax}} \left(\frac{g\left(\mathcal{A}_j, \mathcal{O}_j^{-1}\right)}{\left|\mathcal{O}_j^{-1}\right|} \right), & \text{if } g \text{ is a maximization metric,} \\ \underset{\mathcal{A}_j \in \mathcal{F}}{\operatorname{argmin}} \left(\frac{g\left(\mathcal{A}_j, \mathcal{O}_j^{-1}\right)}{\left|\mathcal{O}_j^{-1}\right|} \right), & \text{if } g \text{ is a minimization metric,} \end{cases} \tag{24}$$

where

$$\mathcal{O}_j^{-1} = \mathcal{O} \setminus \mathcal{O}_j. \tag{25}$$

The evaluation on the individual validation sets $\mathcal{O}_j^{-1}$ ensures that the selected automaton maximizes the consistency on the complete learning set $\mathcal{O}$.

The second inference strategy, the *Majority Vote*-based approach, considers all the automata. For every character in the output trace, we determine the character that is predicted by the majority of the automata in the forest. This is done by a voting scheme, which is formally defined as follows

$$h_{\mathrm{MV}}(\boldsymbol{i}) = \left[\underset{o \in O}{\operatorname{argmax}} \left(\sum_{j=1}^{m} \boldsymbol{o}_j[k] \equiv o \right) \right] \quad \text{with } \boldsymbol{o}_j = \mathcal{A}_j(\boldsymbol{i}), \quad k = 1, ..., |\boldsymbol{i}|, \tag{26}$$

where $\boldsymbol{o}_j[k] \equiv o$ is a boolean function that is 1 if the k-th character of the output trace $\boldsymbol{o}_j$ of automaton $\mathcal{A}_j$ is equal to the character o and 0 otherwise.

Further parameters in the construction of automata forests are:

- $m \in \mathbb{N}$, the number of automata in the forest and
- $\rho \in (0, 1]$, the fraction of the original learning set used to construct an automaton.

Listing 2 outlines the methodology for constructing automata forests. Subsets are collected and automata, forming the instances are learned. The method *createSubset* draws subsets of size $\rho \cdot |\mathcal{O}|$. A learning set $\mathcal{O}$ for DFAs consists of positive and negative examples. The positive and negative subsets are drawn separately to ensure that the instances are learned from a balanced set of positive and negative examples. All subsets

are drawn independently of each other. Thus, learning sets of instances may overlap. Afterward, state merging algorithms build automata for all subsets. Here, we use the RPNI algorithm [123]. Finally, the automata forest $\mathcal{F}$ consists of the set $\{\mathcal{A}^1, ..., \mathcal{A}^m\}$ of automata instances.

4.5.2 Discussion & Properties

Traditional automata learning algorithms guarantee that a learned automaton is consistent on the learning set, i.e., when inferring the learned automaton with one of the inputs from the learning set, the output is the same as the one in the learning set. A consequence of this is that the learned automaton is equivalent to the target automaton when the learning set is a characteristic set as described in Section 3.2.1. We analyze the consistency of automata forests and their convergence to the target automaton.

Theorem 4.1: An automata forest with performance metrics is consistent on the learning set, if and only if the performance metric for the selected automaton $\mathcal{A}_{\text{PM}}$ is optimal.

Proof: We know that every automaton $\mathcal{A}_j$ in the forest is consistent on its learning set $\mathcal{O}_j$ [79]. Further, we know that the performance metric for the automaton $\mathcal{A}_{\text{PM}}$ is optimal, i.e., $g\big(\mathcal{A}_{\text{PM}}, \mathcal{O}_j^{-1}\big)$ is maximal or minimal, depending on the metric. By definition of the performance metric, this means that the automaton $\mathcal{A}_{\text{PM}}$ is consistent on the learning set $\mathcal{O}_j^{-1}$, i.e., it predicts the same output for every input in the learning set. Thus, the automaton $\mathcal{A}_{\text{PM}}$ is consistent on the learning set $\mathcal{O}$. □

From Theorem 4.1, we can conclude that the automata forest with performance metrics converges to the target automaton if the learning set is a characteristic set and the performance metric for selected automaton $\mathcal{A}_{\text{PM}}$ is optimal.

Theorem 4.2: The selected automaton $\mathcal{A}_{\text{PM}}$ is equivalent to the target automaton, if and only if at least one of the instances $\mathcal{A}_j$ in the automata forest is learned from a characteristic set.

Proof: We know that the automaton $\mathcal{A}_j$ is equivalent to the target automaton as it is learned from a characteristic set [79]. Thus, $\mathcal{A}_j$ is consistent on all traces of the target automaton and especially on its validation set $\mathcal{O}_j^{-1}$. Therefore, the performance metric for $\mathcal{A}_j$ is optimal, i.e., $g\big(\mathcal{A}_j, \mathcal{O}_j^{-1}\big)$ is maximal or minimal, depending on the metric. The automata forest selects either the automaton $\mathcal{A}_j$ or an automaton $\mathcal{A}_k$ whose performance metric is also optimal. $\mathcal{A}_k$ is also consistent on the full learning set as the performance metric for $\mathcal{A}_k$ is optimal. We know that the full learning set contains a characteristic set as $\mathcal{O}_j$ is a characteristic set. Thus, $\mathcal{A}_k$ is consistent

on a characteristic set and therefore equivalent to the target automaton. Thus, the automata forest converges to the target automaton. □

Theorem 4.2 shows that the automata forest with performance metrics converges to the target automaton if at least one of the instances in the forest is learned from a characteristic set. Thus, if the full learning set contains a characteristic set, the probability of sampling a characteristic set for at least one of the instances in the automata forest is equivalent to the probability of convergence of the automata forest to the target automaton.

Theorem 4.3: The probability that the automata forest with majority vote is consistent on the learning set is

$$P = \left(\sum_{j=\lceil \frac{m}{2} \rceil}^{m} \binom{m}{j} \rho^j (1-\rho)^{m-j} \right)^{|\mathcal{O}|} \tag{27}$$

for sufficiently large m and ρ.

Proof: We know that the automata forest with majority vote is consistent on the learning set, if and only if the majority of the instances in the forest predict the same output for every input in the learning set. This is the case if at least $\lceil \frac{m}{2} \rceil$ instances predict the same correct output. An instance is guaranteed to predict the correct output for a specific trace $\boldsymbol{s}$ if this trace is contained in the learning set of the instance. The probability that a trace $\boldsymbol{s}$ is contained in the learning set of an instance is ρ. For m independent sub-samples $\mathcal{O}_j$, the number of sub-samples containing $\boldsymbol{s}$ is distributed according to a binomial distribution with parameters m and ρ. Let $X_{\boldsymbol{s}}$ be the random variable representing the number of instances in the automata forest that contain the trace $\boldsymbol{s}$. We seek the probability that $X_{\boldsymbol{s}} \geq \lceil \frac{m}{2} \rceil$, i.e., at least half of the instances contain the trace $\boldsymbol{s}$, which is

$$P_{\boldsymbol{s}} = P\left(X_{\boldsymbol{s}} \geq \left\lceil \frac{m}{2} \right\rceil \right) = \sum_{j=\lceil \frac{m}{2} \rceil}^{m} \binom{m}{j} \rho^j (1-\rho)^{m-j}. \tag{28}$$

Assuming independence in the sampling of traces, which is valid for sufficiently large m and ρ, the probability that this holds for all traces in the learning set is

$$P = (P_{\boldsymbol{s}})^{|\mathcal{O}|} = \left(\sum_{j=\lceil \frac{m}{2} \rceil}^{m} \binom{m}{j} \rho^j (1-\rho)^{m-j} \right)^{|\mathcal{O}|}. \tag{29}$$

□

For the automata forest with majority vote, we conclude that the probability of consistency on the learning set is high if the number of instances m and the fraction of learning data ρ are sufficiently large.

The automata forest with majority vote does not result in a single automaton. Thus, we cannot explicitly apply the convergence theorem of traditional automata learning algorithms. However, we have a probabilistic consistency of the automata forest with majority vote with the following theorem.

Theorem 4.4: If the learning set is a characteristic set, Theorem 4.3 gives the probability of consistency of the automata forest with majority vote with respect to the target automaton.

Proof: The automata forest with majority vote is a deterministic oracle for the same input traces as the target automaton. Thus, the automata forest has an underlying automaton representation $\mathcal{A}_{\text{hidden}}$ that represents the language encoded by the majority vote. We know that the automata forest is consistent on the learning set with the probability given in Theorem 4.3. If the learning set is a characteristic set, the automata forest with majority vote is consistent on this characteristic set. We conclude that the automaton $\mathcal{A}_{\text{hidden}}$ is consistent on the characteristic set and, thus, equivalent to the target automaton. □

In practice, the automata forest with majority vote predicts the correct output for many observations in the learning set, even if only a few instances are consistent on the traces in $\mathcal{O}$ as the other instances do not necessarily reach sufficient votes on false predictions in the majority vote of Equation 26.

Though theoretical guarantees for consistency and convergence for automata forests are not as strong as for traditional automata learning algorithms, the automata forests are a good alternative for learning automata from a learning set that is not necessarily characteristic. The main advantage of the automata forests is that a more thorough exploration of solutions takes place. Given a learning set that is not a characteristic set, there exists a set of multiple automata that are consistent with this set, i.e., could reproduce the same observations, where the actual target automaton is one of them. As automata forests learn multiple automata on smaller subsets of the learning set, they explore the solution space more thoroughly than traditional automata learning algorithms. This idea is illustrated in Figure 39. The black dot represents the target automaton. The red area is the set of automata that are consistent on the learning set where the red dot represents the automaton learned with a traditional automata learning algorithm. The blue area is the set of automata that are consistent on the subsets of the learning set of size ρ. This area is larger than the red area, i.e., the automata forest explores a larger solution space. The blue dots represent the instances in the automata forest. One of the blue dots is considered to be closer to the target automaton than the red dot. The goal of automata forests is to identify and select such better approximations of the target automaton.

This discussion is a conceptual view. We provide an empirical evaluation of automata forests in the next section. The evaluation shows that indeed a better performance is

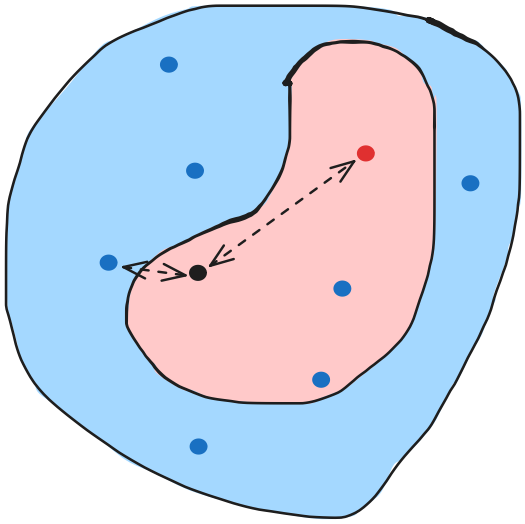


Figure 39 — Concept of automata forests: the black dot represents the target automaton, the blue dots represent the instances in the automata forest and the red dot represents the automaton that is determined with a traditional automata learning algorithm.

achieved with automata forests compared to traditional automata learning algorithms on the considered examples.

4.5.3 Empirical Results

Setup

Examples:

- Coffee: coffee machine example from Section 3.7.1
- Mealy 10: artificial Mealy machine with 10 states, 4 inputs, and 4 outputs
- Mealy 15: artificial Mealy machine with 15 states, 4 inputs, and 4 outputs
- Mealy 20: artificial Mealy machine with 20 states, 4 inputs, and 4 outputs
- Mealy 25: artificial Mealy machine with 25 states, 5 inputs, and 5 outputs

Language: Java

Learning Set:

- Coffee: 1,000 traces
- All others: 10,000 traces

Test Set: 1,000 repetitions with each 1,000 traces for all examples

Metrics:

- Count: number of predicted traces of the model that are equivalent to the ground truth trace
- Hamming: Hamming distance between ground truth and traces predicted by the model

- Welch's t-test p-value for the significance in performance improvements of automata forests compared to traditional automata learning algorithms [140]

Device: IntelCore i7-9750H CPU@2.60GHz, 16GB RAM

We present a performance analysis of automata forests compared to traditional automata learning. Figure 40 to Figure 44 show histograms of the number of correctly translated traces for the coffee machine example and the artificial Mealy machines with 10, 15, 20, and 25 states. The histograms show that the forest with majority vote ($\text{Forest}_{\text{MV}}$) outperforms the other approaches with respect to the number of correctly translated traces. Additionally, the variance of the automata forests is lower than the variance of the RPNI algorithm. This indicates that the automata forests are more robust than the RPNI algorithm by exploring the solution space more thoroughly.

Table VIII provides the average μ of the number of traces from the test set of size 1000 that were correctly translated by the learned model. All forest variants show a better average performance than the RPNI algorithm, where the best performance is achieved with $\text{Forest}_{\text{MV}}$. The results of the Welch's t-test [140] performed with the SciPy package [141] confirm that the improvements are statistically significant in most cases with significance level $p < 0.05$. For the forest with majority voting, the p-value is low for all the examples, indicating consistent statistically significant improvements. The learning time of the automata forests is significantly higher than the learning

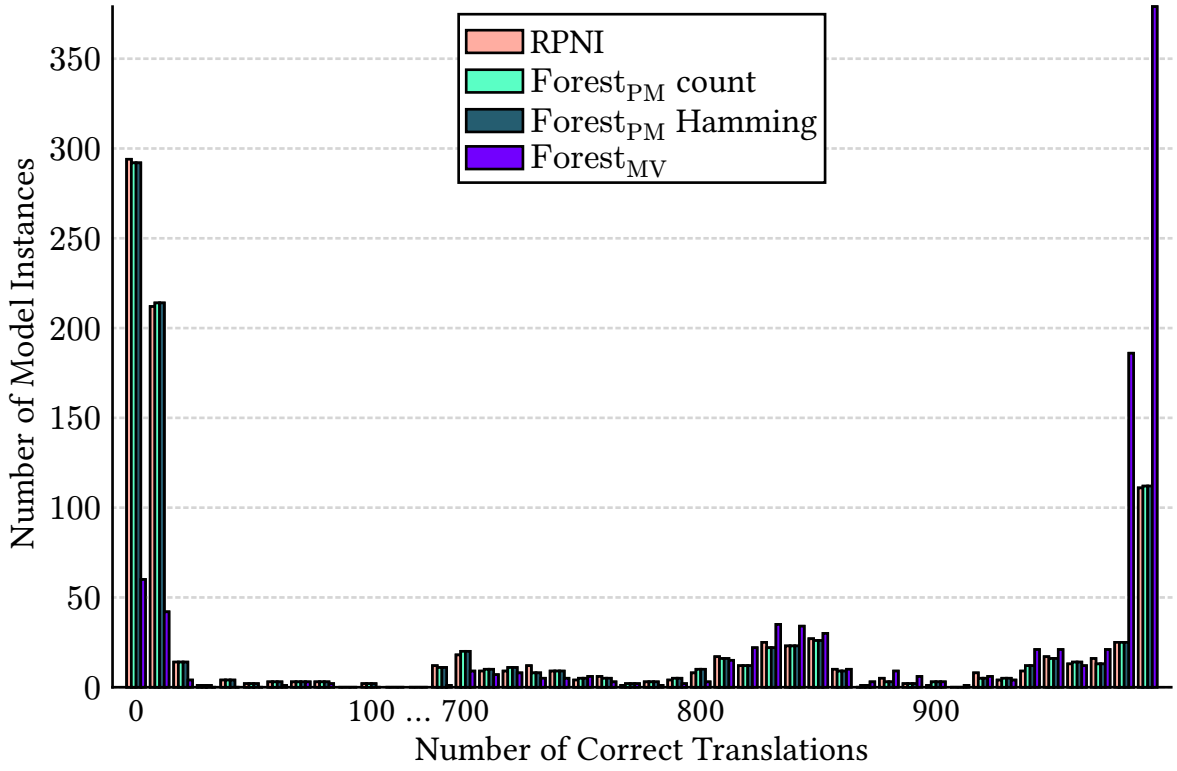


Figure 40 — Histogram over 1000 repetitions of the number of correctly translated traces for the coffee machine example on 1000 trials.

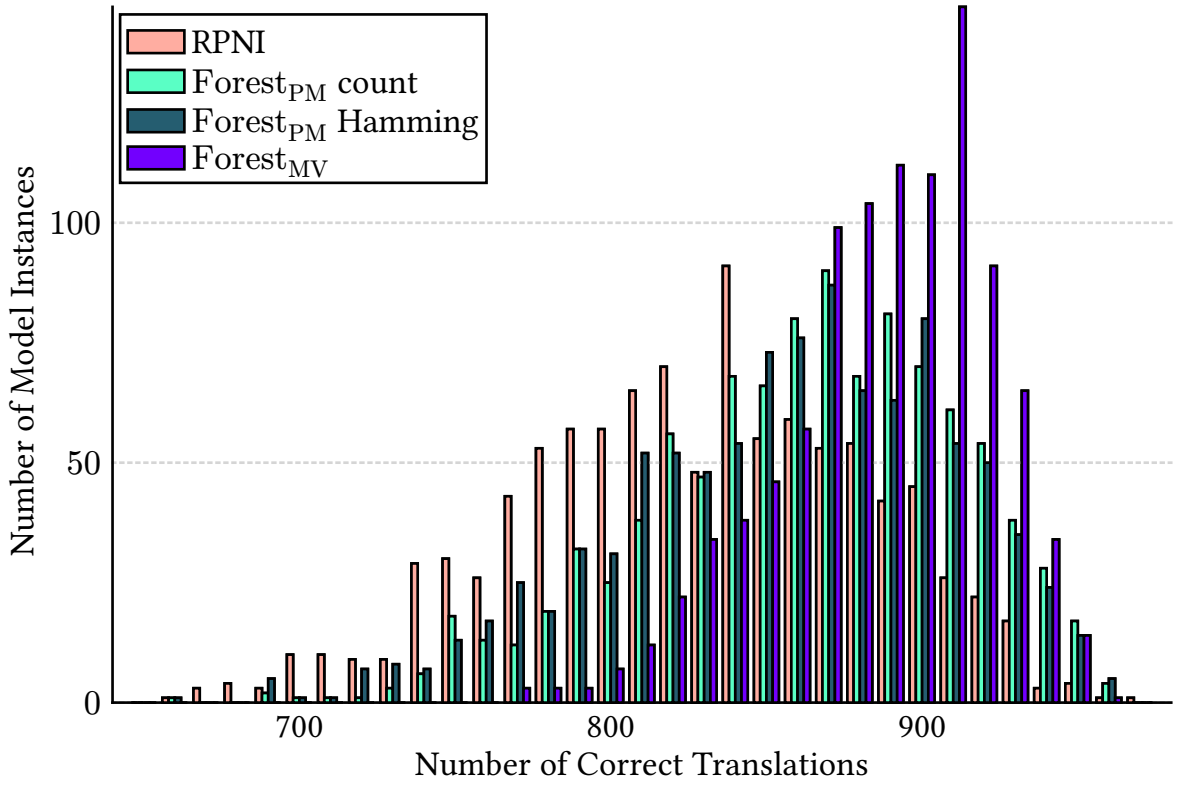


Figure 41 — Histogram over 1000 repetitions of the number of correctly translated traces for the artificial Mealy machine with 10 states on 1000 trials.

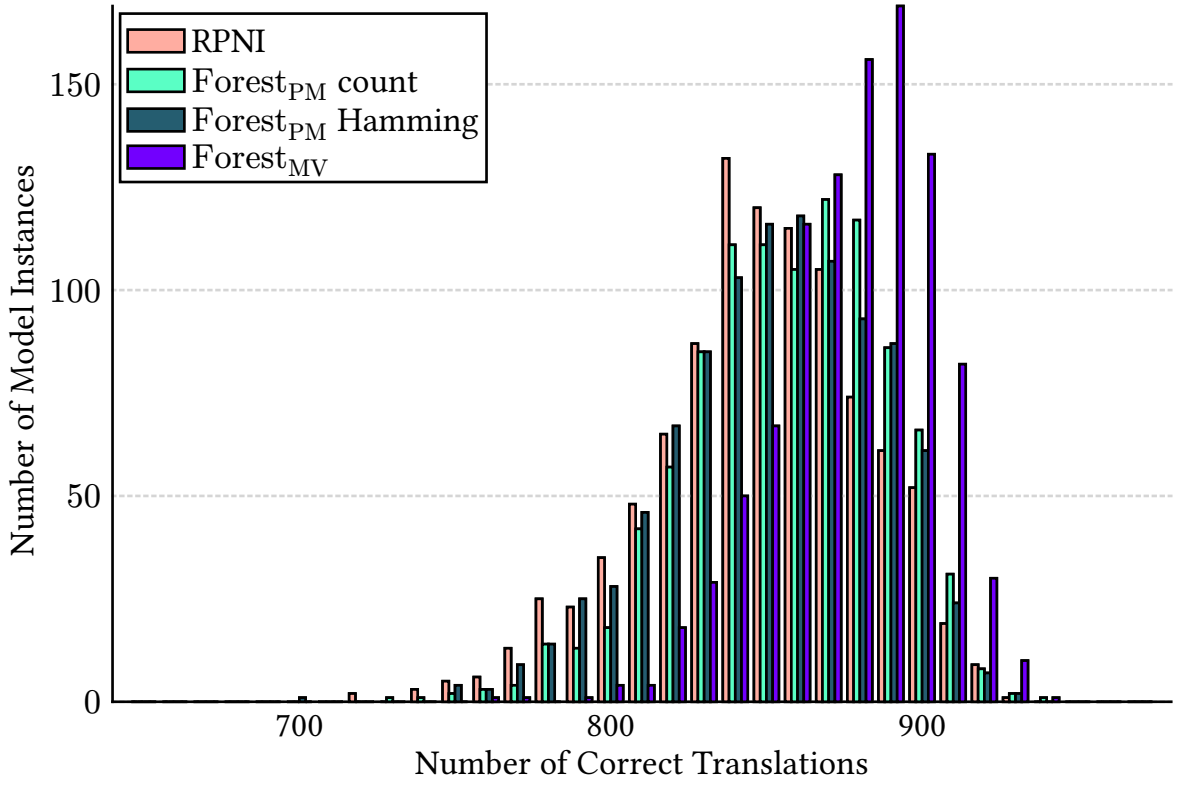


Figure 42 — Histogram over 1000 repetitions of the number of correctly translated traces for the artificial Mealy machine with 15 states on 1000 trials.

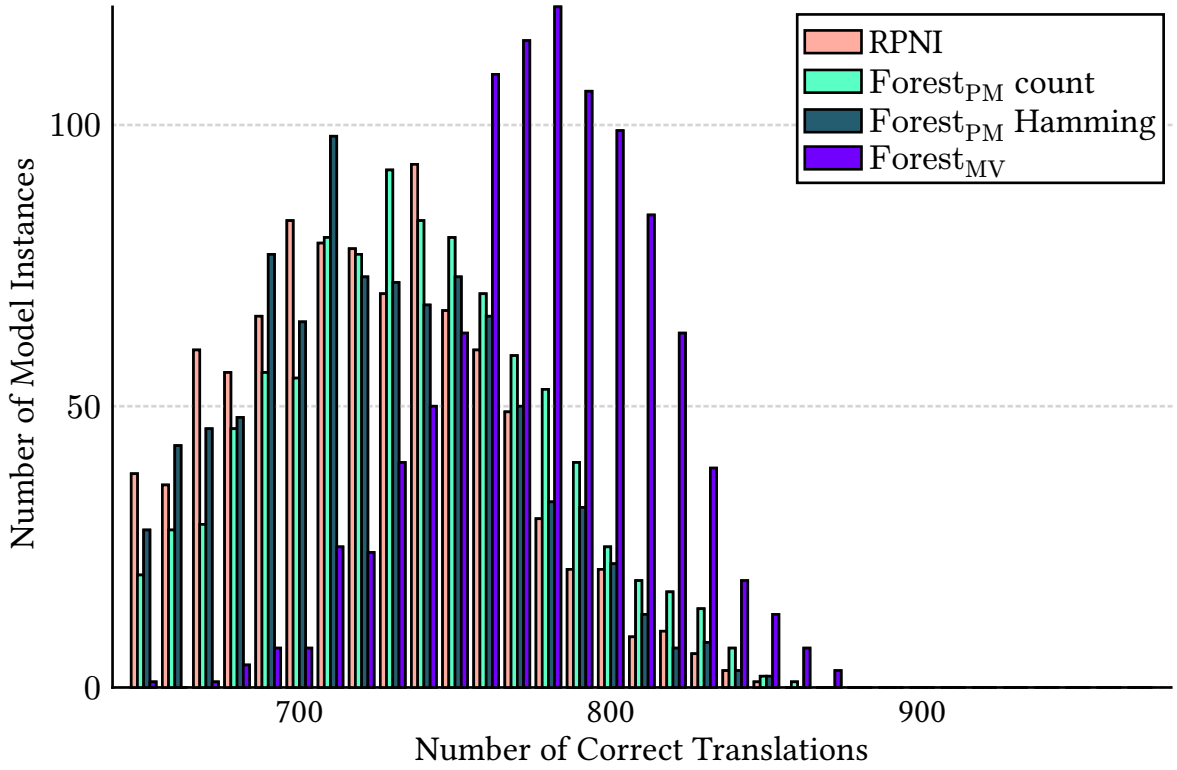


Figure 43 — Histogram over 1000 repetitions of the number of correctly translated traces for the artificial Mealy machine with 20 states on 1000 trials.

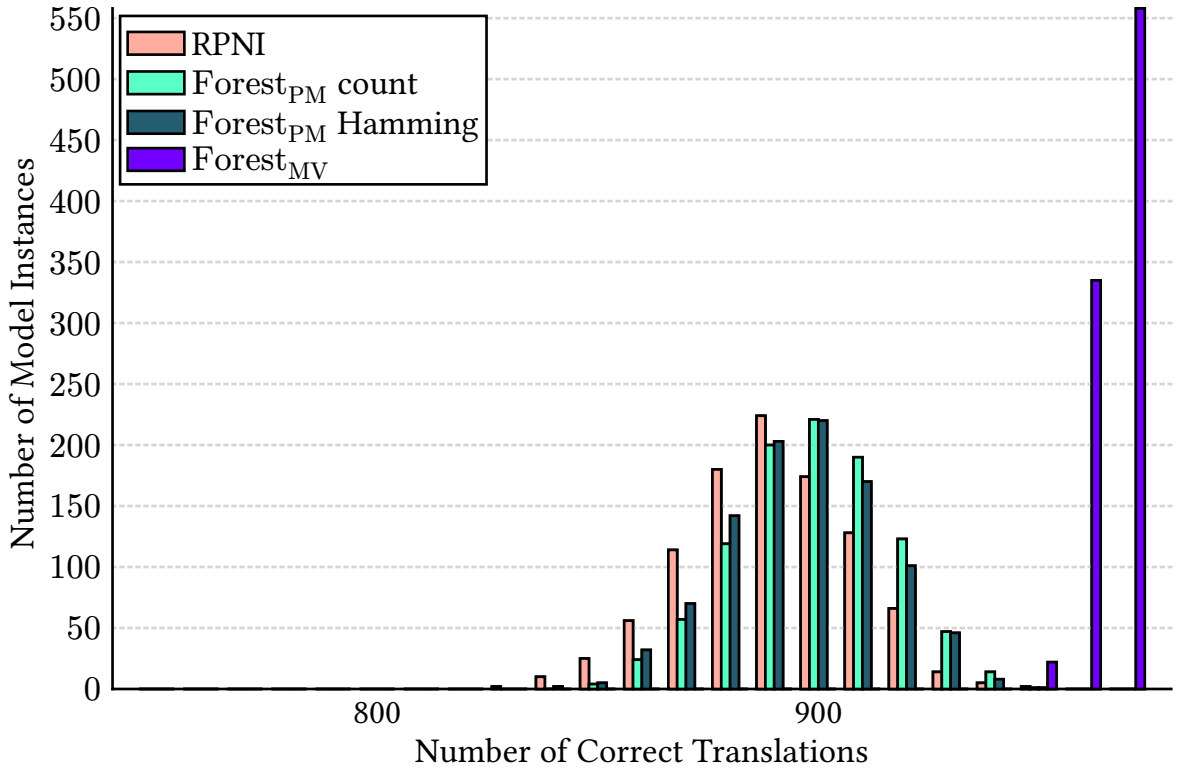


Figure 44 — Histogram over 1000 repetitions of the number of correctly translated traces for the artificial Mealy machine with 25 states on 1000 trials.

Table VIII — Empirical results for the analysis of automata forests. The presented metric is the average μ of the number of traces from the test set of size 1000 that were correctly translated by the learned model, the overall execution time t in seconds and the p-value of the Welch's t-test for the hypothesis that the samples of the automata forests have a larger mean value, are given. The columns show the results for the traditional RPNI, the automata forest with performance metrics ($\text{Forest}_{\text{PM}}$ count), where the count of correctly translated traces is maximized, the automata forest with performance metrics ($\text{Forest}_{\text{PM}}$ Hamming), where the Hamming distance is minimized, and for the automata forest with majority vote ($\text{Forest}_{\text{MV}}$).

Example	RPNI		$\text{Forest}_{\text{PM}}$ count			$\text{Forest}_{\text{PM}}$ Hamming			$\text{Forest}_{\text{MV}}$		
	μ	t [s]	μ	t [s]	p	μ	t [s]	p	μ	t [s]	p
Coffee	395	0.001	394	0.033	0.516	394	0.033	0.516	828	0.033	0.0
Mealy 10	831	0.009	865	0.841	0.0	860	0.841	0.0	892	0.841	0.0
Mealy 15	852	0.008	861	0.739	0.0	857	0.739	0.001	882	0.739	0.0
Mealy 20	720	0.014	734	1.388	0.0	722	1.388	0.236	783	1.388	0.0
Mealy 25	894	0.135	903	20.285	0.0	901	20.285	0.0	972	10.285	0.0

time of the RPNI algorithm. This is due to the fact that multiple automata are learned subsequently. Nevertheless, this could be improved by parallelizing the learning of the automata as the individual automata are learned independently of each other.

4.6 Summary

The results presented in this chapter answer the research question **Q1 — In how far are classical automata learning algorithms applicable to CPSs?** through a systematic investigation of the four sub-questions identified at the beginning of this chapter.

Q1.1 — How to abstract a CPS such that it is learnable by automata learning? We establish that abstraction represents the fundamental prerequisite for successful automata learning of CPSs. Our analysis demonstrates that continuous signals must be discretized, e.g., through domain partitioning, while preserving essential behavioral information. The critical challenge lies in ensuring deterministic observations — the same input trace must consistently produce the same output trace. We show that abstraction choices directly influence the manageability of learned automata size and the preservation of relevant system dynamics.

Q1.2 — Which chances and limitations of active and passive automata learning for CPSs exist? Our comparative analysis shows that both, passive and active, paradigms are applicable to CPSs, but with distinct trade-offs and abstraction require-

ments. Passive automata learning operates on pre-collected traces but requires complete behavioral coverage and consistent initial states — conditions often challenging to ensure in real-world CPSs. Active automata learning enables targeted exploration through direct system interaction but demands accessibility and controllability of the SUL. The paradigm choice influences the required abstraction strategy and practical feasibility.

Q1.3 — How do the learned models support applications such as test case generation? We demonstrate the practical utility of learned automata through systematic test case generation for an RTLS. Our coverage-guided approach transforms abstract automata models into executable test cases, enabling comprehensive system evaluation. This application validates the interpretability and actionability of discrete automata models for CPS analysis.

Q1.4 — How to improve the applicability of automata learning on limited data sets and black-box systems? We propose a novel passive learning strategy called automata forests. Our automata forests approach addresses the fundamental challenge of learning from potentially incomplete datasets. By exploring the solution space more thoroughly through multiple automata instances, forests achieve superior performance compared to traditional algorithms and enhance robustness. The majority vote mechanism particularly excels, demonstrating that automata forests mitigate the limitations of individual learners when characteristic sets are not guaranteed.

Overall Assessment: Classical automata learning algorithms are applicable to CPSs, but with practical limitations. Success requires careful abstraction design, appropriate paradigm selection based on system characteristics, and recognition of accessibility or controllability constraints. While automata provide powerful tools for representing discrete, deterministic behavior, their applicability to real-world CPSs involves theoretical and practical constraints that necessitate the exploration of more flexible modeling approaches — motivating our progression to DTL in the subsequent chapter.

Chapter 5

Decision Tree Models

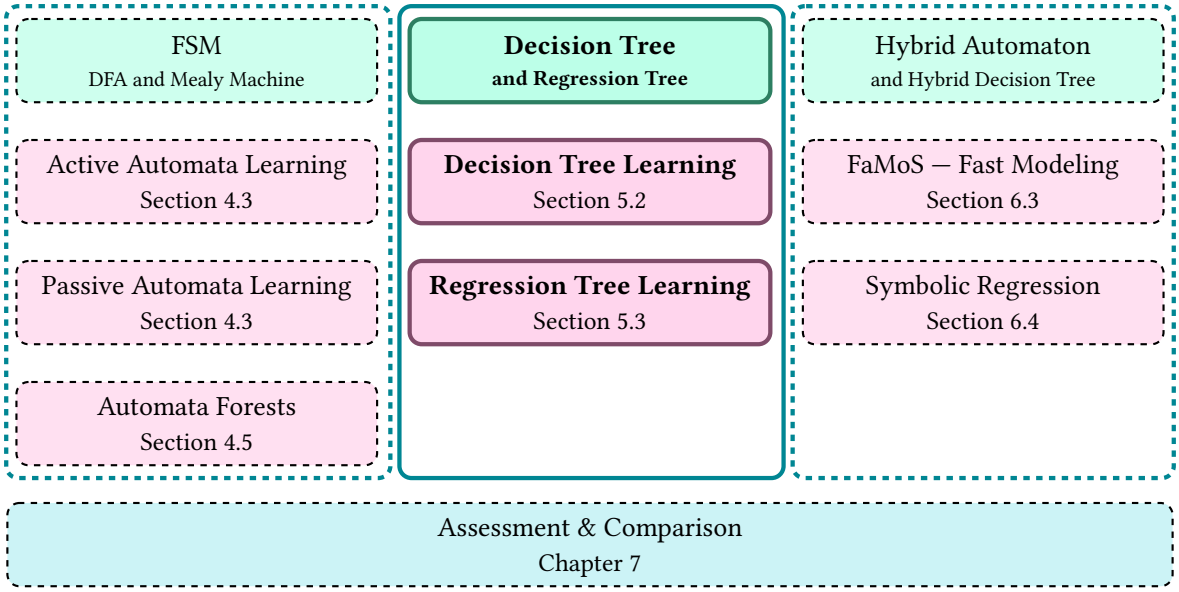


Figure 45 — Overview of the presented models (green) and learning approaches (pink) per model type in this work. This chapter presents decision tree models for CPS.

The limitations of deterministic discrete models from the previous chapter motivate the exploration of more flexible discrete modeling approaches. While automata learning has a strong theoretical foundation, the requirement for deterministic behavior severely constrains the applicability to real-world CPSs where environmental

uncertainty and measurement noise introduce non-deterministic observations. As demonstrated in our LiDAR case study, the same observed environmental configuration can produce varying localization errors, violating the deterministic assumption fundamental to finite state machine representations.

In this chapter, we investigate DTL as an alternative discrete modeling approach that addresses the core limitations of automata learning while maintaining interpretability. Figure 45 positions this thesis within our modeling framework, representing the transition from purely deterministic automaton models to decision trees with temporal information. Decision trees naturally handle non-deterministic behavior by learning mappings from input histories to output predictions, thus overcoming the primary constraint of automata learning.

Our investigation extends classical decision tree applications by integrating temporal information to capture the sequential nature of CPS behavior. This temporal integration addresses a fundamental challenge in CPS modeling: systems exhibit memory effects where past states influence current outputs, necessitating models that can represent such dependencies. We address the second research question **Q2 — What is the performance of decision trees that integrate temporal information for modeling of CPSs?** through the following sub-questions:

- Q2.1 — How to integrate temporal information into decision trees for modeling CPSs?
- Q2.2 — How do decision tree models with temporal information relate to FSMs?
- Q2.3 — What is the quantitative approximation of decision tree models compared to the original system?
- Q2.4 — How do regression trees contribute to the modeling of systems with continuous behavior?
- Q2.5 — How do decision tree models support applications such as test case generation?

We begin by developing a novel preprocessing methodology that transforms temporal observations into feature vectors suitable for DTL, enabling the integration of bounded history information while maintaining computational tractability. This preprocessing aligns the presentations by S. Plambeck et al. [30] [31] and addresses the fundamental challenge of learning from non-reset observations — a critical practical advantage over automata learning, where system resets are often impossible in operational CPSs. The bounded history approach provides a principled solution to the infinite memory problem while introducing controlled approximation that we quantify theoretically.

Subsequently, we establish formal connections between decision tree models and FSMs. This theoretical analysis reveals fundamental limitations of learning from finite observations, establishing that decision tree models form supersets of the original system languages. We quantify these approximation bounds and identify the precise conditions under which exact reconstruction becomes possible. Compared to our previous publications S. Plambeck et al. [30] and S. Plambeck et al. [31], we extend the empirical evaluation to additional examples and provide comprehensive comparisons

with neural network models. The practical validation of our approach encompasses both discrete and continuous modeling scenarios. We demonstrate monitoring capabilities that leverage decision trees as uncertainty indicators to detect anomalous behavior.

Finally, we showcase the practical applicability through novel test case generation strategies presented by S. Plambeck et al. [33]. We exploit decision tree models to achieve comprehensive system coverage using an intermediate FSM representation of the decision tree.

The progression through these contributions establishes decision trees as a powerful intermediate representation between deterministic automata and fully continuous models, setting the stage for the hybrid automata approaches explored in the subsequent chapter.

5.1 Related Work

Tree structures are already established in the context of discrete models and automata learning, e.g., as prefix trees in the RPNI algorithm [79] or active automata learning approaches such as the TTT algorithm [124]. Even though decision trees are usually used for classification, there exists other work considering decision trees on temporal data. E. Lucena-Sanchez et al. [142] discuss feature selection strategies for decision and regression tree learning on time-series data. Another approach learns decision trees on temporal data via interval logic [15]. D. Della Monica et al. [23] use decision trees on temporal logic to classify time series data. Finally, L. Goupil et al. [143] use decision trees in combination with symbolic classification for diagnosis of CPSs.

In the scope of PAC learning, M. J. Kearns [76] discuss also the identification of automata from observations without reset. While traditional automata learning requires a reset of the system to observe the system from the initial state, M. J. Kearns [76] propose a method to learn automata from observations without reset. The idea is the identification of a so called homing sequence to determine the current state in the automaton. Nevertheless, M. J. Kearns [76] show that the identification of a homing sequence is, in general, computationally expensive. Similar approaches for automata learning without reset have been proposed, e.g., by G.-V. Jourdan et al. [144] and R. Groz et al. [145]. These approaches require or generate prior knowledge of the system to distinguish and characterize traces of the states of the system. With DTL, we do not require or search for homing sequences. Instead, we assume a limited length of the observed traces and analyze the uncertainty resulting from this assumption.

Furthermore, previous approaches for the approximation of automata models exist, e.g., regarding fuzzy models [146] and for the reduction of the size of an automaton [147]. We provide a novel and parallel approach in approximate models for CPSs using decision trees.

5.2 Discrete Decision Tree Models

S. Plambeck et al. [30] introduce two types of decision tree models for representing systems with discrete Mealy machine behavior. The first model determines whether a given trace is valid for the system. Within this context, we derive theoretical limitations for learning Mealy machine models from observations with bounded history. The second model uses a decision tree to predict the next output based on a history of inputs and outputs. This predictive model is evaluated empirically in a monitoring scenario to detect faulty or unexpected system behavior.

5.2.1 Observations & Decision Tree Learning

We consider a system with a Mealy machine representation $\mathcal{A}_M$ as introduced in Definition 3.5. This Mealy machine representation provides the necessary abstraction of the system to discrete behavior and defines the language L of possible traces that start in the initial state of the system.

We observe a set of traces $\mathcal{O}_{\text{trace}} \subseteq \Sigma^n$ on the system, where $\Sigma = I \times O$ is the input-output alphabet according to Section 3.2 of $\mathcal{A}_M$ and n is the length of the traces. In addition to a trace $\boldsymbol{s} \in \mathcal{O}_{\text{trace}}$ with every element in the alphabet $\Sigma = I \times O$, we consider the sub-traces $\boldsymbol{o} \in O^n$ and $\boldsymbol{i} \in I^n$ as the output and input traces, respectively. In contrast to a learning set for automata learning, the traces in $\mathcal{O}_{\text{trace}}$ do not have to start in the initial state of the system, instead, they have a bounded length n. DTL requires a set of observations $\mathcal{O}_{\text{DTL}}$ that consists of feature vectors and class labels. To construct $\mathcal{O}_{\text{DTL}}$ from $\mathcal{O}_{\text{trace}} \subseteq \Sigma^n$, we classify a trace $\boldsymbol{s} \in \mathcal{O}_{\text{trace}}$ with respect to the language L of the Mealy machine as defined in Chapter 3 as follows

$$d_L(\boldsymbol{s}) = \begin{cases} 1, & \text{if } \exists \boldsymbol{r} \in L \text{ and } \boldsymbol{t}, \boldsymbol{u} \in \Sigma^* \text{ where } \boldsymbol{tsu} = \boldsymbol{r}, \\ 0, & \text{otherwise,} \end{cases} \tag{30}$$

or with respect to the output as follows

$$d_O(\boldsymbol{s}[1, ..., n-1], \boldsymbol{i}[n]) = \boldsymbol{o}[n], \tag{31}$$

where $\boldsymbol{i}[n]$ and $\boldsymbol{o}[n]$ are the last input and output symbol of the trace $\boldsymbol{s}$, respectively.

We then convert the set of traces $\mathcal{O}_{\text{trace}}$ into a set of observations for DTL using the trace $\boldsymbol{s}$ as the feature vector and the result of the classification function d_L or d_O as the class label:

$$\begin{gathered} \mathcal{O}_{\text{DTL}}^L = \{\langle \boldsymbol{s}, d_L(\boldsymbol{s}) \rangle : \boldsymbol{s} \in \mathcal{O}_{\text{trace}}\} \\ \text{or} \\ \mathcal{O}_{\text{DTL}}^O = \{\langle [\boldsymbol{s}[1, ..., n-1], \boldsymbol{i}[n]], d_O(\boldsymbol{s}[1, ..., n-1], \boldsymbol{i}[n]) \rangle : \boldsymbol{s} \in \mathcal{O}_{\text{trace}}\}. \end{gathered} \tag{32}$$

We define *complete* learning sets for both cases as follows

- the complete language-based set holds all sequences of length n on the system:

$$\mathcal{O}_n^L = \mathcal{O}_{\Sigma^n}^L = \{\langle \boldsymbol{s}, d_L(\boldsymbol{s}) \rangle : \boldsymbol{s} \in \Sigma^n\}, \tag{33}$$

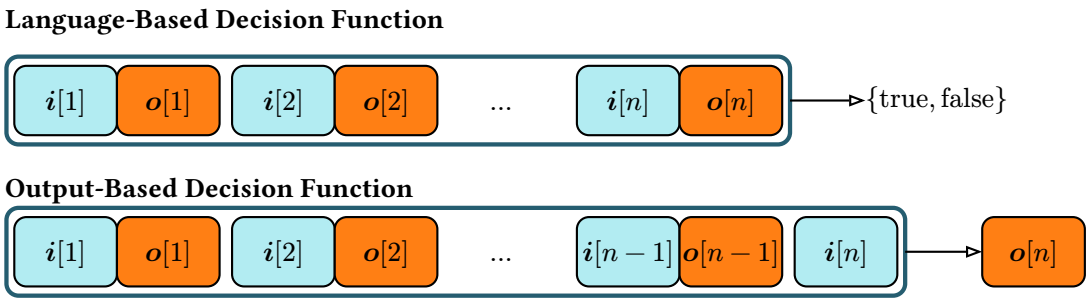


Figure 46 — Illustration of the decision functions for language-based and output-based decision tree models.

- the complete output-based set is the set of all substrings of length n of the words in the language L of the Mealy machine $\mathcal{A}_M$:

$$\mathcal{O}_n^O = \left\{ \boldsymbol{s} \in \Sigma^n : \exists \boldsymbol{r} \in L \text{ and } \boldsymbol{t}, \boldsymbol{u} \in \Sigma^* \text{ where } \boldsymbol{tsu} = \boldsymbol{r} \right\}. \tag{34}$$

Example 8: For the coffee machine example from Figure 11, the complete sets of observations for $n = 2$ are

$$\begin{aligned} \mathcal{O}_n^L = \{ & \langle [\texttt{CLEAN|ok}, \texttt{CLEAN|ok}], 1 \rangle, \\ & \langle [\texttt{CLEAN|ok}, \texttt{CLEAN|error}], 0 \rangle \\ & \ldots, \\ & \langle [\texttt{POD|ok}, \texttt{CLEAN|ok}], 1 \rangle, \\ & \langle [\texttt{POD|ok}, \texttt{CLEAN|error}], 0 \rangle, \\ & \langle [\texttt{POD|ok}, \texttt{POD|ok}], 1 \rangle, \ldots \} \end{aligned} \tag{35}$$

and

$$\begin{aligned} \mathcal{O}_n^O = \{ & [\texttt{CLEAN|ok}, \texttt{CLEAN|ok}], \\ & \ldots, \\ & [\texttt{POD|ok}, \texttt{CLEAN|ok}], \\ & [\texttt{POD|ok}, \texttt{POD|ok}], \ldots \}. \end{aligned} \tag{36}$$

Using the abstractions of traces to the sets of observations $\mathcal{O}_{\text{DTL}}^L$ or $\mathcal{O}_{\text{DTL}}^O$, we learn a decision tree model $\mathcal{T}_L$ or $\mathcal{T}_O$. For black-box systems, where the Mealy machine representation is not available, the set of observations $\mathcal{O}_{\text{DTL}}$ is usually not complete. For theoretical considerations and baseline evaluations on systems with known models, we employ the complete sets of observations $\mathcal{O}_n$. The decision tree model represents the system behavior on a bounded history of length n. Figure 46 illustrates the decision function of the language-based decision tree model and the output-based decision tree model. Given a bounded history of length n

- $\mathcal{T}_L$ determines whether a trace is valid for the system or
- $\mathcal{T}_O$ predicts the next output symbol based on the last n input-output pairs.

```
1:  function CLASSIFY(s, 𝒯_L, n)
2:      d ← GETCLASSIFICATIONFUNCTION(𝒯_L)
3:      if |s| < n then
4:          return true
5:      end
6:      for i = 1...|s| − n do
7:          if ¬d(s[i, i + n]) then
8:              return false
9:          end
10:     end
11:     return true
12: end
```

Listing 3 — Classification of a trace $\boldsymbol{s}$ using a decision tree model with language-based binary classification function d.

In the following, we analyze properties and limitations of the language-based model $\mathcal{T}_L$. Afterward, we extend the analysis to the output-based model $\mathcal{T}_O$ and derive a bound on the prediction error of the model. Finally, we evaluate the decision tree models in a monitoring scenario.

5.2.2 Language-Based Model & Decision Tree Language

We determine whether the language-based decision tree model is an exact representation of the original Mealy machine, i.e., whether the language L of the Mealy machine is represented in the decision tree. For this, we define a language L_D for the decision tree model.

We use Listing 3 to classify a trace $\boldsymbol{s}$ of arbitrary length with the decision tree model. By this, we define the language L_D as the set of all traces that are valid according to the classification by Listing 3. According to Listing 3, a trace $\boldsymbol{s}$ is classified as valid if the length is smaller than n or if all its sub-traces of length n are valid on the decision tree model. Formally, we define L_D as

$$L_D = \{\boldsymbol{s} \in \Sigma^* : \textsc{classify}(\boldsymbol{s}, \mathcal{T}_L, n)\}. \tag{37}$$

To understand the best-case performance of DTL, the following analysis considers a decision tree that is learned on $\mathcal{O}_n^L$, i.e., the set of all possible observations. We show that a full equivalence on the languages L of the Mealy machine and L_D of the decision tree model is only achieved if restrictions apply to the language L. In the general case, the language L_D of the decision tree model is a superset of the original language L, i.e., $L \subseteq L_D$.

Theorem 5.1: When learning a decision tree from the observations $\mathcal{O}_n^L$ then the learned language L_D is a superset of the original language L, i.e., $L \subseteq L_D$.

Proof: An equivalent formulation of $L \subseteq L_D$ is

$$\boldsymbol{s} \in L \Rightarrow \boldsymbol{s} \in L_D \quad \text{for} \quad \boldsymbol{s} \in \Sigma^*. \tag{38}$$

Any trace of length shorter or equal to n is in L_D, because the decision tree model classifies all traces of length shorter or equal to n as valid. Thus, we only have to consider traces of length greater than n in the following. From the definition of $\mathcal{O}_n^L$, we know that all traces of length n are in $\mathcal{O}_n^L$ and are classified according to Equation 30. Thus, the following holds

$$\boldsymbol{s} \in L \Rightarrow \langle \boldsymbol{s}[j, j+n], 1 \rangle \in \mathcal{O}_n^L, \forall j \in \{1, 2, ..., |\boldsymbol{s}| - n\}. \tag{39}$$

Further, the construction of the decision tree model $\mathcal{T}_L$ from the observations $\mathcal{O}_n^L$ leads to the relation

$$\langle \boldsymbol{s}[j, j+n], 1 \rangle \in \mathcal{O}_n^L \Rightarrow d^L(\boldsymbol{s}[j, j+n]) \equiv 1, \forall j \in \{1, 2, ..., |\boldsymbol{s}| - n\}, \tag{40}$$

where d^L is the decision function of the learned decision tree. Equation 40 holds, because $\mathcal{O}_n^L$ is a complete set and every feature vector appears only once in $\mathcal{O}_n^L$. A misclassification of any feature vector would indicate that the same feature appears with a different output label in $\mathcal{O}_n^L$, which is a contradiction to the definition of $\mathcal{O}_n^L$. From Listing 3, we conclude

$$\begin{gathered} d^L(\boldsymbol{s}[j, j+n]) \equiv 1, \forall j \in \{1, 2, ..., |\boldsymbol{s}| - n\} \\ \Leftrightarrow \\ \textsc{classify}(\boldsymbol{s}, \mathcal{T}_L, n) \equiv \text{true}. \end{gathered} \tag{41}$$

From Equation 37, we know that

$$\textsc{classify}(\boldsymbol{s}, \mathcal{T}_L, n) \equiv \text{true} \Leftrightarrow \boldsymbol{s} \in L_D. \tag{42}$$

Equation 39 to Equation 42 directly lead to the relation from Equation 38, showing that the language of the decision tree is a superset of the language of the system, i.e., $L \subseteq L_D$.

□

To achieve full equivalence of the two languages L and L_D, we need the additional relation

$$\boldsymbol{s} \in L \Leftarrow \boldsymbol{s} \in L_D \quad \text{for} \quad \boldsymbol{s} \in \Sigma^*. \tag{43}$$

An immediate counterexample is that every trace of length less than or equal to n is included in L_D, while only a subset of these traces belong to L. Moreover, we can construct further counterexamples to disprove Equation 43 for traces longer than n. According to Listing 3 and Equation 37, all traces in L_D are composed of valid subtraces from $\mathcal{O}_n^L$. However, it is possible to construct a trace from valid sub-traces in $\mathcal{O}_n^L$ that does not actually belong to L.

	i	o
A	x	a
B	y	a
C	x	b
D	y	b
E	x	c
F	y	c

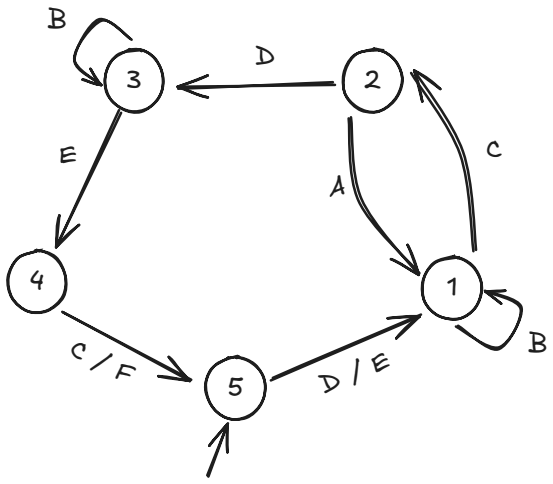


Figure 47 — Mealy machine with alphabet $\Sigma = \{A, B, C, D, E, F\}$. The mapping to input-output pairs is given in the table on the left. The system is not learnable from bounded observations because of the two self-loops on the same symbol B.

Example 9: We consider the Mealy machine from Figure 47, which has an input alphabet of size two and an output alphabet of size three. We define the alphabet as $\Sigma = I \times O = A, B, C, D, E, F$. The Mealy machine in Figure 47 has two self-loops corresponding to the same symbol from Σ. We find the traces $[A\ C\ D\ B\ B\ B\ B\ B\ E\ B]$ and $[A\ B\ B\ B\ B\ B\ C\ D]$, which are in the language L of the system. If we choose $n = 5$, the valid traces of length n are:

$$\{[A\ C\ D\ B\ B], [C\ D\ B\ B\ B], [D\ B\ B\ B\ B], [B\ B\ B\ B\ B], \\ [B\ B\ B\ B\ E], [A\ B\ B\ B\ B], [B\ B\ B\ B\ C]\}. \tag{44}$$

The trace $[A\ C\ D\ B\ B\ B\ B\ B\ C]$, according to Listing 3, would belong to the language L_D. However, the given system cannot produce this trace and, thus, is not in the language L.

In conclusion, no exact reconstruction of a general regular language L of a Mealy machine from the decision tree model is possible. Example 9 shows a counterexample for the reconstruction of the language L from observations of bounded history n. This example is agnostic to the usage of decision trees for the construction of a model, but is a general problem of learning from bounded history.

Definition 5.1: An *ambiguous observation* is a pair of observations $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ of length n that are valid traces of the Mealy machine $\mathcal{A}_M$, i.e., $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathcal{O}_n^O$ and the two traces are equivalent except for the last output symbol, i.e.,

$$\begin{aligned} \boldsymbol{s}_1[j] &\equiv \boldsymbol{s}_2[j] \quad \forall j \in \{1, 2, ..., n-1\}, \\ \boldsymbol{i}_1[n] &\equiv \boldsymbol{i}_2[n], \\ \boldsymbol{o}_1[n] &\neq \boldsymbol{o}_2[n]. \end{aligned} \tag{45}$$

Definition 5.1 describes the ambiguity of observations that occurs when learning from bounded history of length n. In contrast to non-deterministic behavior, which is inherent ambiguity of a system, the ambiguity of observations is a result of the bounded history as the original Mealy machine representation of the system is deterministic. In the following, we analyze the effects of these ambiguities on the decision tree model. We use the output-based decision tree model $\mathcal{T}_O$ to predict the next output symbol based on the last n input-output pairs and analyze the prediction accuracy of the model.

5.2.3 Output-Based Model & Error Bounds

We switch to the output-based decision tree model $\mathcal{T}_O$ to analyze the prediction of the next output symbol based on the last n input-output pairs. Impure leaves of the decision tree model $\mathcal{T}_O$ indicate the existence of ambiguous observations in the original Mealy machine representation $\mathcal{A}_M$. Impure leaves are leaves of the decision tree model that contain more than one class label, i.e.,

$$\begin{aligned} V_a &= \{v \in V_L : |C_v| > 1\} \\ \text{with} \quad C_v &= \{c \in C : \exists \langle \boldsymbol{f}, c \rangle \in \mathcal{O}_v\}, \end{aligned} \tag{46}$$

using the notation from Section 3.1.

Theorem 5.2: The output-based decision tree model $\mathcal{T}_O$ that is learned on the set $\mathcal{O}_n^O$ has an impure leaf v_a if and only if there exists an ambiguous observation according to Definition 5.1 for the Mealy machine representation $\mathcal{A}_M$ and a finite history of length n.

Proof: An impure leaf $v_a \in V_a$ contains more than one class label, i.e.,

$$v_a \in V_a \Leftrightarrow \left|C_{v_a}\right| > 1. \tag{47}$$

Thus, there exist two observations $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ with the same feature vector $\boldsymbol{f}$, but different class labels $c_1 \neq c_2$:

$$\left|C_{v_a}\right| > 1 \Leftrightarrow \exists \langle \boldsymbol{f}, c_1 \rangle, \langle \boldsymbol{f}, c_2 \rangle \in \mathcal{O}_{v_a} \text{ with } c_1 \neq c_2. \tag{48}$$

From the construction of the decision tree model $\mathcal{T}_O$ in Equation 31, we know that both observations are in the set $\mathcal{O}_n^O$, i.e., $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathcal{O}_n^O$, i.e.,

$$\exists v \in V_L \text{ with } \langle \boldsymbol{f}, c_1 \rangle, \langle \boldsymbol{f}, c_2 \rangle \in \mathcal{O}_v \Leftrightarrow \langle \boldsymbol{f}, c_1 \rangle, \langle \boldsymbol{f}, c_2 \rangle \in \mathcal{O}_n^O. \tag{49}$$

Thus, both observations are valid traces of the Mealy machine representation $\mathcal{A}_M$. From Equation 48, we know that the two traces are equivalent except for the last output symbol, i.e., $\boldsymbol{o}_1[n] = c_1 \neq c_2 = \boldsymbol{o}_2[n]$. Thus, $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are ambiguous observations according to Definition 5.1. □

We further analyze the prediction error of the decision tree model $\mathcal{T}_O$ with additional knowledge on the abstraction of observations of the system to discrete traces. We assume that the abstraction of the system to a Mealy machine representation $\mathcal{A}_M$

bases on a discretization of the system inputs and outputs as described in Section 4.2 and that the system is observed without noise. For the evolvement of the system over time as well as the categorization subspaces of the outputs, we assume a bound on the maximum distance between two consecutive outputs in the discrete space:

$$\begin{gathered} \operatorname{dist}\left(o_j, o_{j+1}\right) \leq \lambda \\ \text{with} \quad o_j = q_O(\boldsymbol{o}_c[t]), o_{j+1} = q_O(\boldsymbol{o}_c[t+t_S]) \\ \forall t \in \mathbb{R}, \end{gathered} \tag{50}$$

where *dist* is a distance function on the discrete space, $\lambda \in \mathbb{R}$ is a constant, and t_S a fixed sampling period that satisfies the Nyquist-Shannon sampling theorem. Based on these assumptions, we derive a bound on the prediction error between the predicted output o_{pred} and the actual output o_{obs} of the system taking into account the effects of sampling, discretization, and ambiguous observations:

$$e = \operatorname{dist}\left(o_{\text{pred}}, o_{\text{obs}}\right). \tag{51}$$

Theorem 5.3: The maximum error of a decision tree model learned on the set $\mathcal{O}_n^O$ prediction is $\epsilon_\infty = 2 \cdot \lambda$.

Proof: Assume, we are given a pair of ambiguous observations $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$. From Definition 5.1, we know that

$$\begin{aligned} &\boldsymbol{s}_1[j] \equiv \boldsymbol{s}_2[j] \quad \forall j \in \{1, 2, ..., n-1\}, \\ &\boldsymbol{i}_1[n] \equiv \boldsymbol{i}_2[n], \\ &\boldsymbol{o}_1[n] \neq \boldsymbol{o}_2[n]. \end{aligned} \tag{52}$$

In the learning set $\mathcal{O}_n^O$, the two traces form observations with the same feature vector

$$\boldsymbol{f} = [\boldsymbol{s}_1[1, 2, ..., n-1], \boldsymbol{i}_1[n]] \equiv [\boldsymbol{s}_2[1, 2, ..., n-1], \boldsymbol{i}_2[n]], \tag{53}$$

but different class labels

$$c_1 = \boldsymbol{o}_1[n] \text{ and } c_2 = \boldsymbol{o}_2[n]. \tag{54}$$

As the decision tree model $\mathcal{T}_O$ is learned on the set $\mathcal{O}_n^O$, the prediction of the decision tree model for the feature vector $\boldsymbol{f}$ is either c_1 or c_2. Without loss of generality, we assume the prediction of the decision tree for $\boldsymbol{f}$ is c_2. The two observations share an identical previous input-output combination $\langle \boldsymbol{i}[n-1], \boldsymbol{o}[n-1] \rangle$. From Equation 50, we know that the system output changes at most by the distance of λ in the discrete space within a sampling period. We conclude that both class labels c_1 and c_2 have at most a distance of λ to the previous output symbol $\boldsymbol{o}[n-1]$ in the discrete space, i.e.,

$$\operatorname{dist}(\boldsymbol{o}[n-1], c_1) \leq \lambda \quad \text{and} \quad \operatorname{dist}(\boldsymbol{o}[n-1], c_2) \leq \lambda. \tag{55}$$

Thus, the distance of the class label at most adds up to their distance to the previous output symbol $\boldsymbol{o}[n-1]$ in the discrete space, i.e.,

$$\operatorname{dist}(c_1, c_2) \leq 2 \cdot \lambda. \tag{56}$$

□

Based on the learned decision tree model, we determine a tighter estimation ϵ_{pract} for the worst case prediction error.

Theorem 5.4: The maximum error of a decision tree prediction learned on the set $\mathcal{O}_n^O$ is bounded by

$$\epsilon_{\text{pract}} = \max_{v \in V_L} \{ \text{dist}(c_i, c_j) : c_i, c_j \in C_v \}. \tag{57}$$

Proof: The formulation of ϵ_{pract} determines the maximum distance between two class labels that both come from the same leaf v of the decision tree model $\mathcal{T}_O$. We assume an observation $\langle \boldsymbol{f}, c_1 \rangle$ with feature vector $\boldsymbol{f}$ and class label c_1, thus,

$$\exists v \in V_L \text{ with } \langle \boldsymbol{f}, c_1 \rangle \in \mathcal{O}_v. \tag{58}$$

Further, we assume that $c_2 = d(\boldsymbol{f}) \neq c_1$. Thus, another observation $\langle \boldsymbol{f}, c_2 \rangle$ is also in the set $\mathcal{O}_v$, i.e.,

$$\exists v \in V_L \text{ with } \langle \boldsymbol{f}, c_2 \rangle \in \mathcal{O}_v. \tag{59}$$

We conclude that

$$c_1, c_2 \in C_v \tag{60}$$

and

$$\text{dist}(c_1, c_2) \leq \epsilon_{\text{pract}}. \tag{61}$$

□

The value ϵ_{pract} is only a strict upper bound, if the decision tree model is learned on the complete set $\mathcal{O}_n^O$. In a practical scenario, e.g., for black-box systems, $\mathcal{O}_n^O$ is usually not available. In this case, the ϵ_{pract} is not a strict upper bound, but an estimation of the maximum prediction error. An upper bound for ϵ_{pract} is again given by $2 \cdot \lambda$. Further, ϵ_{pract} is determined by the decision tree model $\mathcal{T}_O$. Thus, the decision tree model does not only serve for a prediction of a next timestep, but provides further information on the dynamics of the system.

5.2.4 Prediction Model & Monitoring

Setup

Examples:

- Water Tank: flushing and discrete variant of the water tank from Section 3.7.2
- Coffee: coffee machine from Section 3.7.1
- Boiler: boiler system from Section 3.7.4 in the controlled temperature range variant. The output signal, i.e., the temperature of the boiler, is discretized to a set of three values representing the intervals $[0, 20]$, $(20, 23]$, and $(23, 26]$.
- Ambiguous: Mealy machine with ambiguous behavior as given in Figure 47

Language: Matlab, Java
Scenarios:
1) Complete scenario: all possible observations $\mathcal{O}_n^O$ are available for learning. The set $\mathcal{O}_n^O$ is only feasible for small transition systems. Thus, this scenario is only evaluated on the water tank and the ambiguous example.
2) Practical scenario: a fixed amount of data is available for learning.

Learning Set:
- Complete scenario: $\mathcal{O}_n^O$ with all possible observations of length n. The amount of observations depends on the size of the alphabet $|\Sigma|$ and the length of the observations n.
- Practical scenario: 200 random sequences of length between 10 and 50 starting in the initial state. For DTL and neural network learning, all subsequences of length n are extracted from the sequences. For automata learning, the sequences are used as they are.

Test Set:
- Complete scenario: $\mathcal{O}_n^O$ with all possible observations of length n. This is the complete set of possible samples.
- Practical scenario: 50 positive traces. The positive traces are enriched with negative traces by altering the last output symbol in each subsequence of length n.

Metrics:
- True Positives: prediction of an output identical to the positive observation
- False Negatives: prediction of an output different from the positive observation
- False Positives: prediction of an output identical to the negative observation
- True Negatives: prediction of an output different from the negative observation
- Accuracy: percentage of correct predictions, i.e., true positives and true negatives

Device: IntelCore i7-9750H CPU@2.60GHz, 16GB RAM

We evaluate the prediction accuracy of our decision tree models. These results directly serve as an indicator for the usability of decision tree models in a monitoring scenario and provide insights in the theoretical findings of the previous sections for learning decision tree models from observations of bounded history.

For our evaluation, the decision tree predicts the next output of the system based on the last n steps. Thus, running the model in parallel to the original system implements a monitor that detects an unexpected or erroneous behavior of the system online, whenever the predicted output differs from the real output.

We evaluate the prediction on a test set, which contains observations of correct system behavior and wrong, i.e., erroneous system behavior, which should be detected

by a monitor. We analyze the prediction accuracy of the decision tree model as well as the false positives and true positives. The number of false positives is crucial as it defines the cases where an error is not detected. The true positives indicate the cases where the decision tree model correctly predicts the next output of the system.

As given in the setup, we evaluate two scenarios: the complete scenario and the practical scenario. In the complete scenario, we compare our decision tree models to neural networks. The two scenarios differ in the learning set. In the complete scenario, we use the complete set of observations $\mathcal{O}_n^O$ for learning. In the practical scenario, we use a fixed amount of data for learning, where the completeness of the learning set is not guaranteed.

5.2.4.1 Complete Scenario

In the complete scenario, we compare the decision tree model to a neural network model. The architecture of the neural network is shown in Figure 48. The neural network has an input layer with n neurons, where n is the length of the bounded history. The next layer is a fully connected hidden layer with ten neurons followed by a ReLU activation layer before the second fully connected layer. In this layer, the number of neurons is equivalent to the number of classes in the output. The next layer is a softmax activation layer before the final output layer providing the predicted class label. The neural network is trained with the `fitcnet` function of Matlab [148], which uses a limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newton algorithm (LBFGS) [149] as its loss function minimization technique minimizing the cross-entropy loss.

The results of the complete scenario are shown in Table IX. This scenario showcases the capabilities of the decision tree model in detecting ambiguous observations as well as the prediction accuracy of the model. The table gives the accuracy for the decision tree model (T) and the neural network (NN) for different lengths of bounded history n. Furthermore, the elapsed learning time for both models is given. For complete learning data, the performance of the neural network is identical to the performance of the decision tree model. Still, the learning time for the decision tree model is always lower than the learning time for the neural network. The number of nodes in the decision tree is given in the table as $|V|$ and the number of neurons in the neural network is given by the column v_{NN}.

With an increasing length of bounded history n, the prediction accuracy increases. For the discrete water tank example, 100% accuracy is reached for $n = 2$.

A key insight, as established in Theorem 5.2, is that impure leaves in the decision tree model correspond exactly to ambiguous observations arising from the bounded history n. The number of impure leaves $|V_a|$, thus, quantifies the inherent ambiguity present in the system for the chosen history length. According to the theoretical results, perfect prediction accuracy (100%) is only achievable when no such ambiguities exist for the given n — that is, when all leaves are pure.

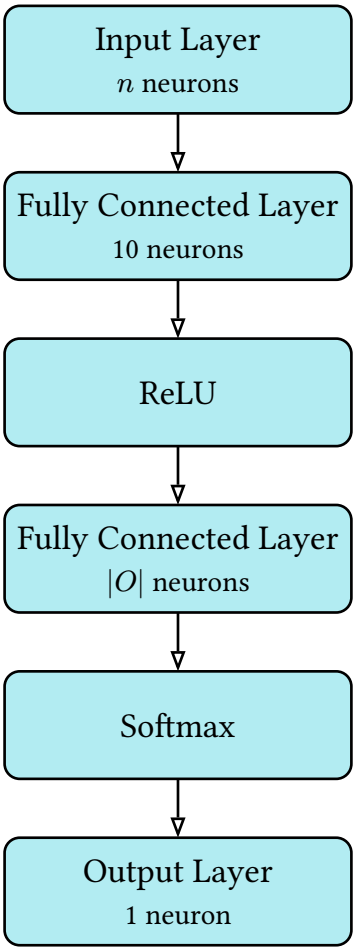


Figure 48 — Architecture of the neural network used for comparison with the decision tree model. The input layer has n neurons, where n is the length of the bounded history. The output layer has one neuron for each possible output symbol of the system.

The ambiguous example from Figure 47 has two loops with the output symbol B such that ambiguous observations for this example exist for all window sizes $n \in \mathbb{N}^+$. Thus, the example illustrates the challenges posed by ambiguous observations. Consequently, the number of impure leaves remains constant as n increases, and the prediction accuracy is bounded away from 100%, in accordance with the error bounds derived earlier. These empirical results are a consequence of the theoretical findings: the structure of the decision tree — specifically, the presence and number of impure leaves — directly reflects the fundamental limitations imposed by learning from bounded history.

5.2.4.2 Practical Scenario

In the practical scenario, i.e., when applying the models to data sets of fixed size, we compare the decision tree model with both a neural network and an automaton model. The neural network architecture and training procedure are identical to those used in the complete scenario. The neural networks and decision trees are trained on the same

Table IX — Prediction accuracy of decision tree models (T) and neural networks (NN) on complete learning sets for different n. The number of nodes in the decision tree is given as $|V|$ and the number of neurons in the neural network is given by the column v_{NN}. The number of impure leaves is given as $|V_a|$. The learning time in seconds is given as t_{learn}.

Example	n	$\lvert V_a\rvert$	$\lvert V\rvert$	v_{NN}	Accuracy in %		t_{learn} [s]	
					NN	*T*	*NN*	*T*
water tank	2	0	13	17	100.00	100.00	0.03	0.01
ambiguous	2	2	7	16	80.00	80.00	0.04	0.01
ambiguous	3	2	13	17	90.00	90.00	0.03	0.01
ambiguous	5	2	27	18	97.50	97.50	0.05	0.01
ambiguous	6	2	33	19	98.75	98.75	0.08	0.01
ambiguous	8	2	45	20	99.69	99.69	0.12	0.01
ambiguous	11	2	63	31	99.96	99.96	0.91	0.03

data. However, a direct comparison with the automaton model is not straightforward, as automata learning requires training samples that all start from the initial state of the original system. To address this, we preprocess the data differently for each model: the automaton model is trained directly on sequences starting in the initial state. For decision trees and neural networks, we extract all subsequences of length n from these sequences to form the learning set. As a result, the automaton model benefits from access to longer, uninterrupted sequences, potentially providing more information than the other two models receive.

Consequently, the evaluation of the automaton model requires a different approach than the decision tree and neural network models. We evaluate the automaton model on the same test set of observations of bounded history n as the decision tree and neural network models. For prediction, we use the automaton model to search for the occurrence of the last $n-1$ samples in a path of the automaton. The next output is then predicted based on the transition for the next input symbol that follows this path in the automaton.

The results of the practical scenario are shown in Table X. This scenario showcases the practical applicability of the decision tree model in monitoring scenarios. The table gives the percentage of false positive and true positive predictions for the decision tree model (DT), the neural network model (NN), and the automaton model (A) for different lengths of bounded history n.

Similar to the complete scenario, the performance of the decision tree and the neural network is nearly identical. The number of nodes in the decision tree is similar to the number of neurons in the neural network; however, the decision tree is trained in a shorter time. A significant increase in the number of nodes for the decision tree is only observed in the boiler example, where the model exceeds 600 nodes. In this case, the

Table X — Prediction accuracy in terms of false positive and true positive predictions of decision tree models (T) and neural networks (NN) and automaton models (A) on practical learning sets for different n. The number of nodes in the decision tree is given as $|V|$ and the number of neurons in the neural network is given by the column v_{NN}. The number of ambiguous leaves is given as $|V_a|$. The learning time in seconds is given as t_{learn}.

Example	n	$\lvert V_a\rvert$	$\lvert V\rvert$	$\lvert Q\rvert$	v_{NN}	False Pos. in %			True Pos. in %			t_{learn} [s]		
						A	*T*	*NN*	*A*	*T*	*NN*	*A*	*T*	*NN*
ambiguous	2	4	11	5	16	9.83	7.09	7.09	83.48	87.56	87.56	0.06	0.25	1.32
water tank discrete	2	0	11	4	17	0.00	0.00	0.00	100.00	100.00	100.00	0.06	0.03	0.26
coffee	2	1	9	6	16	0.20	0.20	0.20	99.40	99.40	99.40	0.06	0.06	0.50
boiler $t_S = 10$	2	21	45	579	16	6.13	0.57	0.57	49.88	95.46	95.45	1.07	1.40	18.55
water tank $t_S = 1$	2	2	7	-	17	-	8.20	8.00	-	75.02	75.02	-	0.02	0.18
ambiguous	4	4	29	5	18	1.51	1.65	1.65	96.49	96.56	96.56	0.06	0.02	0.59
coffee	4	1	21	6	18	0.00	0.00	0.00	100.00	100.00	100.00	0.06	0.02	0.22
water tank $t_S = 0.1$	4	4	7	46	18	1.96	1.96	1.96	93.96	93.96	93.96	0.14	0.15	4.03
water tank $t_S = 0.5$	4	2	7	10	19	3.30	3.22	3.22	89.69	89.69	89.69	0.10	0.03	0.57
water tank $t_S = 1$	4	0	7	-	19	-	0.00	0.00	-	100.00	100.00	-	0.02	0.08
water tank $t_S = 1.5$	4	2	9	-	19	-	4.27	4.27	-	88.80	88.80	-	0.02	0.37
ambiguous	5	4	31	5	19	1.12	0.89	0.89	97.70	97.99	97.99	0.06	0.02	0.46
coffee	5	0	33	6	19	0.00	0.00	0.22	100.00	100.00	99.78	0.06	0.04	0.15
ambiguous	6	4	43	5	19	0.77	0.46	0.46	98.69	99.00	99.00	0.06	0.03	1.11
coffee	6	0	29	6	20	0.00	0.08	0.15	100.00	99.92	99.85	0.06	0.02	0.16
ambiguous	8	2	47	5	22	0.08	0.25	0.25	99.58	99.50	99.50	0.06	0.02	0.28
coffee	8	0	19	6	22	0.00	0.00	0.17	100.00	99.83	99.75	0.06	0.02	0.19
ambiguous	11	1	65	5	25	0.00	0.10	0.10	100.00	99.71	99.81	0.06	0.03	0.39
coffee	11	0	19	6	25	0.00	0.29	0.29	100.00	99.62	99.33	0.06	0.03	0.13
boiler $t_S = 10$	11	113	683	579	25	1.40	0.33	0.34	88.79	97.20	97.18	1.07	0.47	44.32

neural network also requires substantially more training time than the decision tree and all other examples.

Despite being trained on a data set with unbounded traces, the automaton model is outperformed in the presented metrics by both the neural network and decision tree models. This is partly because the automaton model predicts only based on the first occurrence of a sequence and does not distinguish between multiple occurrences, limiting its ability to resolve ambiguities. In contrast, the decision tree selects the most likely outcome in ambiguous situations. Only for larger values of n, where ambiguities are rare, does the automaton model occasionally achieve a higher true positive rate

than the decision tree and the neural network. For the boiler example, the decision tree model outperforms the automaton model by more than 40% in true positive rate for $n = 1$, and still by nearly 10% for $n = 10$.

For the discretized examples, i.e., the boiler and the water tank, automaton models are not always learnable (indicated by a - in Table X). For the continuous models, the discretization might lead to non-deterministic observations, which are not representable in deterministic automata. Here, the decision tree and the neural network are able to generalize the observations and predict a next output symbol also in the presence of non-deterministic behavior.

The advantage of the decision tree model is the ability to indicate the existence of ambiguities. Still, in the presence of practical learning data, i.e., possibly incomplete data sets, there is no guarantee that all ambiguities have been observed. Thus, if the number of detected ambiguities is zero, there might still exist undetected ambiguities. At the same time, any detected ambiguity is a true ambiguity on the system.

5.3 Regression Tree Models

In the previous section, we model continuous systems with decision trees by discretizing the continuous signals. However, this abstraction to discrete values can lead to information loss and reduced accuracy. Additionally, choosing appropriate discretization intervals is challenging and may introduce ambiguities, resulting in impure leaves in the decision tree model. To address these issues, we now consider continuous regression trees, which model continuous systems directly on real-valued signals without the need for prior discretization. Instead, regression trees partition the input space into regions, i.e., automatically applying discretization in form of piecewise constant approximations to the continuous signals.

5.3.1 Regression Tree Modeling

We assume a binary regression tree, where each inner node has exactly two outgoing edges. A regression tree is constructed as introduced in Section 3.1. In contrast to a decision tree, a regression tree represents a regression function d_R.

Regression trees, like many flexible machine learning models, are more susceptible to overfitting than decision trees for classification, since they create very specific splits to fit continuous-valued outputs [20]. To mitigate overfitting, several strategies can be employed, such as limiting the maximum depth or requiring a minimum number of samples per leaf during learning. Another option is to prune the tree after training, i.e., removing branches to reduce the size and complexity of the tree. In our approach, we focus on binning the real-valued features as a form of regularization. Binning groups continuous feature values into a finite number of intervals, effectively reducing the complexity of the feature space and making the model less sensitive to small variations in the data. This binning is an automatic discretization of the continuous features,

which helps to prevent the regression tree from creating overly specific splits, thus improving generalization.

We use equiprobable (equal-frequency) binning, which divides the range of each real-valued feature into intervals containing approximately the same number of samples. This approach is conceptually similar to the discretization used in decision trees for classification, but here the bins are determined automatically from the data distribution. Importantly, while the binning of features helps to prevent the tree from creating overly specific splits, thus improving generalization, the regression tree still predicts continuous outputs.

The learning set $\mathcal{O}_{\text{RTL}}$ for regression trees is constructed from a set of continuous traces $\mathcal{O}^c_{\text{trace}} \subseteq \Sigma^n$ of the system, where each trace $\boldsymbol{s}$ is a sequence of input-output pairs $\boldsymbol{s} = [\langle i_1, o_1\rangle, \langle i_2, o_2\rangle, ..., \langle i_n, o_n\rangle]$. We convert $\mathcal{O}^c_{\text{trace}}$ to $\mathcal{O}_{\text{RTL}}$ consisting of tuples of feature vectors and outputs:

$$\mathcal{O}^O_{\text{RTL}} = \{\langle [i_1, o_1, ..., i_{\text{n-1}}, o_{\text{n-1}}, i_n], o_n\rangle : \boldsymbol{s} \in \mathcal{O}^c_{\text{trace}}\}. \tag{62}$$

The regression tree model predicts a continuous-valued output based on the previous $n-1$ input and output signals as well as the nth input. The prediction accuracy is validated on an evaluation set $\mathcal{O}_E$ consisting of tuples of feature vectors and a known next output $\langle \boldsymbol{f}, o_{\text{exact}}\rangle$. We use the mean absolute error

$$e_{\text{mean}} = \frac{1}{|\mathcal{O}_E|} \sum_{\langle \boldsymbol{f}, o_{\text{exact}}\rangle \in \mathcal{O}_E} |o_{\text{exact}} - d_R(\boldsymbol{f})| \tag{63}$$

and the maximum absolute error

$$e_{\text{max}} = \max_{\langle \boldsymbol{f}, o_{\text{exact}}\rangle \in \mathcal{O}_E} |o_{\text{exact}} - d_R(\boldsymbol{f})| \tag{64}$$

as quality metrics for the learned regression tree model.

5.3.2 Empirical Evaluation

Setup

Examples:

- Water Tank: flushing variant of the water tank from Section 3.7.2
- Boiler: test scenario of the boiler system from Section 3.7.4 with input domain $\{18\,°\text{C}, 19\,°\text{C}, ..., 25\,°\text{C}\}$

Language: Matlab

Learning Set: 200 simulation runs per example

Test Set: 50 simulation runs per example

Metrics:

- Mean absolute error e_{mean},
- Maximum absolute error e_{max}

Table XI — Prediction accuracy of the regression tree model in terms of mean absolute error and maximum absolute error for different n. The number of nodes in the regression tree is given as $|V|$ and the number of neurons in the neural network is given by v_{NN}. The learning time in seconds is given as t_{learn}.

Example	n	$\lvert V\rvert$	v_{NN}	e_{mean}		e_{max}		t_{learn} [s]	
				RT	NN	RT	NN	RT	NN
boiler T=10	2	1233	14	0.10	0.11	0.83	0.11	1.98	0.87
boiler T=20	2	1511	14	0.14	2.32	1.55	2.32	0.21	0.10
boiler T=30	2	1429	14	0.14	0.29	1.31	0.29	0.20	0.12
boiler T=10	7	1091	19	0.02	0.08	0.63	0.08	0.22	16.76
boiler T=20	7	1107	19	0.03	0.16	0.99	0.16	0.06	9.74
boiler T=30	7	1219	19	0.05	0.22	1.30	0.22	0.28	3.77
boiler T=10	10	683	22	0.02	0.13	0.46	0.13	0.21	33.54
boiler T=20	10	1129	22	0.03	0.15	0.89	0.15	0.10	22.56
boiler T=30	10	1253	22	0.04	0.21	1.35	0.21	0.21	4.26
watertank T=0.1	2	163	14	0.03	0.00	0.07	0.00	0.59	37.19
watertank T=0.7	2	165	14	0.03	0.02	0.36	0.02	0.02	2.78
watertank T=1	2	165	14	0.04	0.13	0.51	0.13	0.01	0.26
watertank T=0.1	5	481	17	0.01	0.00	0.03	0.00	0.17	56.89
watertank T=0.7	5	401	17	0.02	0.00	0.06	0.00	0.04	8.65
watertank T=1	5	413	17	0.02	0.00	0.06	0.00	0.03	8.02
watertank T=0.1	10	445	22	0.01	0.00	0.04	0.00	0.31	70.91
watertank T=0.7	10	461	22	0.01	0.00	0.04	0.00	0.03	4.42
watertank T=1	10	423	22	0.01	0.00	0.05	0.00	0.02	1.02

We evaluate the regression tree model for different sampling periods t_S and different lengths of bounded history n. Table XI provides the mean absolute error e_{mean} and maximum absolute error e_{max} of the regression tree model for the boiler example and the water tank for different sampling periods t_S and bounded history n. Further, we compare to the performance of a neural network with the architecture of Figure 49. The neural network is trained with the `fitrnet` function of Matlab [150], which uses a limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newton algorithm (LBFGS) [149] as its loss function minimization technique minimizing the MSE. We observe that for both examples and all sampling periods the mean error decreases over increasing n. This aligns with the results from the previous section, where we observed that the prediction accuracy of decision tree models increases with increasing length of bounded history. Further, the results show a slightly increased prediction accuracy with smaller sampling period, which is an intuitive result.

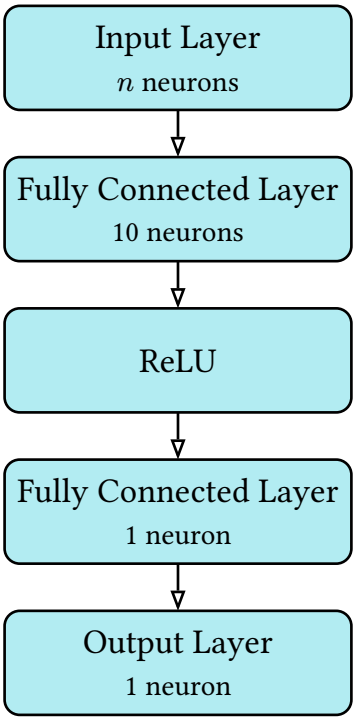


Figure 49 — Neural network architecture for regression tasks. The network consists of an input layer, two hidden layers with ReLU activations, and an output layer.

Furthermore, we observe a stagnation of the error for larger n. For the water tank example, the neural networks achieve a prediction error of zero due to the noise free environment. For the boiler example, the remaining error is primarily caused by sampling effects and the generalization of the regression tree model, which is influenced by the binning of the continuous features. Due to the nature of the simulation, the sampled values of the continuous signals are never exactly identical between runs, as both the simulation speed and parameters (e.g., the target values for the boiler) vary between runs. This leads to small variations in the observed data, even without any noise in the system. As a result, for the shown examples, increasing the bounded history beyond $n = 6$ does not yield further improvements in prediction accuracy.

The performance of the neural network in comparison to the regression tree is similar for both examples. While the error of the neural network is slightly lower for the water tank, the regression tree achieves better results for the boiler example. Further, the regression tree has a larger size of the model when comparing the number of nodes in the tree to the number of neurons in the neural network, but the neural network has a longer learning time.

In summary, regression tree models extend the applicability of DTL to systems with continuous-valued signals, eliminating the need for manual discretization. Regression trees achieve high prediction accuracy while maintaining interpretability and computational efficiency. The empirical results demonstrate that regression trees match or outperform neural networks in terms of mean and maximum prediction error, particularly for non-deterministic and noisy environments. This positions regression trees as

a practical and interpretable modeling approach for continuous CPSs, complementing the discrete decision tree models and providing a foundation for more advanced hybrid modeling techniques in subsequent chapters.

5.4 Test Case Generation with Decision Tree Models

Section 4.4 discusses model-based testing from learned automaton models with coverage-guided test case generation. In the following, we present a novel approach for test case generation using the output-based decision tree models. For this, we analyze the temporal progression of the feature vectors using the decision tree model. Based on that, we derive an illustrative representation of the decision tree model as an FSM, which finally motivates a test coverage metric for decision tree models. For nominal decision trees — those used for classification rather than modeling temporal system behavior — test cases are typically derived by ensuring that each classification (i.e., each leaf) is covered, resulting in one test case per leaf [151]. However, for decision tree models representing temporal system behavior, we present new and more nuanced coverage metrics.

5.4.1 Finite State Machine Representation of the Decision Tree Model

We use output-based decision tree models to represent the behavior of a system. The decision function $d : \Sigma^n \times I \to O$ of the model predicts the output of a next timestep for a feature vector consisting of the history of length n and an input symbol. We introduce an update operator $\ll$ for feature vectors $\boldsymbol{f} = [\langle \boldsymbol{i}[0], \boldsymbol{o}[0] \rangle, \langle \boldsymbol{i}[1], \boldsymbol{o}[1] \rangle, ..., \langle \boldsymbol{i}[n-1], \boldsymbol{o}[n-1] \rangle, \boldsymbol{i}[n]]$, which updates the feature vector by removing the first element and appending a new element at the end, i.e.,

$$\begin{aligned} &[\langle \boldsymbol{i}[0], \boldsymbol{o}[0] \rangle, \langle \boldsymbol{i}[1], \boldsymbol{o}[1] \rangle, ..., \langle \boldsymbol{i}[n-1], \boldsymbol{o}[n-1] \rangle, \boldsymbol{i}[n]] \ll \langle \boldsymbol{o}[n], \boldsymbol{i}[n+1] \rangle \\ &\qquad := [(\boldsymbol{i}[1], \boldsymbol{o}[1]), (\boldsymbol{i}[2], \boldsymbol{o}[2]), ..., (\boldsymbol{i}[n], \boldsymbol{o}[n]), \boldsymbol{i}[n+1]]. \end{aligned} \tag{65}$$

The update of a feature vector describes a transition from one leaf to another leaf of the decision tree model:

- The prediction of a decision tree is the label of the leaf reached when traversing the tree with the feature vector.
- Considering a feature vector $\boldsymbol{f}$ and its updated version $\boldsymbol{f}_{\text{new}}$, these correspond to two leaves l and l_{new} in the decision tree.
- The update of the feature vector $\boldsymbol{f}$ to $\boldsymbol{f}_{\text{new}}$ is a transition from leaf l to leaf l_{new}.

Based on the *leaf transitions*, we construct an FSM representation $\mathcal{A}_{\text{DT}} = (Q, \Sigma, Q_0, \theta)$ of a learned decision tree model to visualize the temporal progression of feature vectors over time. The states of the FSM are defined as follows

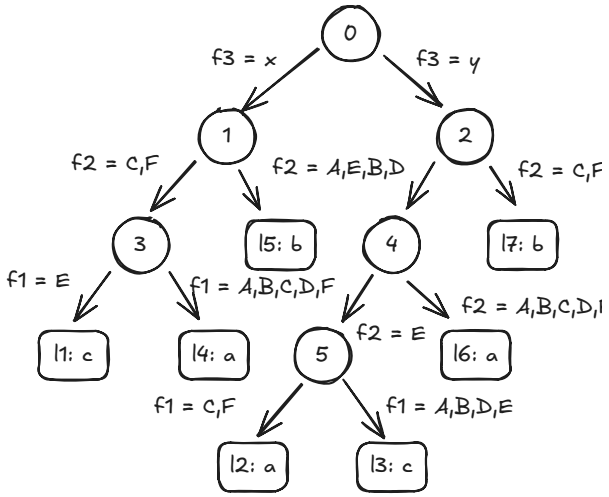


Figure 50 — Decision tree model of the FSM from Figure 47 with $n = 2$.

$$Q = \{l : l \in V_L\}, \tag{66}$$

where V_L is the set of all leaves in the decision tree. The set of initial states is $Q_0 = Q$ and the alphabet is a tuple of input and output symbols $\Sigma = I \times O$. Transitions between states are introduced based on all possible updates of the feature vectors. The input label i of a transition from state l in the FSM representation is an input symbol while the output label is the output of the leaf reached with any updated feature vector, i.e.,

$$\theta(i, l(\boldsymbol{f})) = \{\langle l(\boldsymbol{f} \ll \langle i, d(\boldsymbol{f}, i)\rangle), d(\boldsymbol{f}, i)\rangle : \forall \boldsymbol{f} \in \mathbb{F}\} \tag{67}$$

with $i \in I$ and using the notation $l(\boldsymbol{f})$ for the leaf reached with the feature vector $\boldsymbol{f}$.

The resulting FSM is not necessarily deterministic, because the update of feature vectors from one leaf can lead to different next states for the same input symbol. As discussed previously and shown by Theorem 5.1, the decision tree model of the system is not equivalent to the dynamics of the original system behavior, in case of ambiguous observations. This difference between the decision tree model and the original system behavior is reflected in the FSM representation of the decision tree model as well.

Example 10: Figure 50 illustrates the decision tree model for the example Mealy machine from Figure 47 with $n = 2$ and a complete learning set $\mathcal{O}_2^O$. From this decision tree, we construct the conceptual FSM representation, shown in Figure 51.

We walk through a concrete example of how the feature vectors progress over time. Starting with the feature vector $\boldsymbol{f} = \langle E, C, x\rangle$, we begin in leaf l_1 and yield the output c. If the next input is x, the feature vector updates to $\boldsymbol{f} = \langle C, E, x\rangle$, moving us to leaf l_5. If instead, the next input is y, the feature vector becomes $\boldsymbol{f} = \langle C, E, y\rangle$, leading to leaf l_2. These transitions, highlighted in red in Figure 51, are deterministic because the next state is uniquely determined by the input.

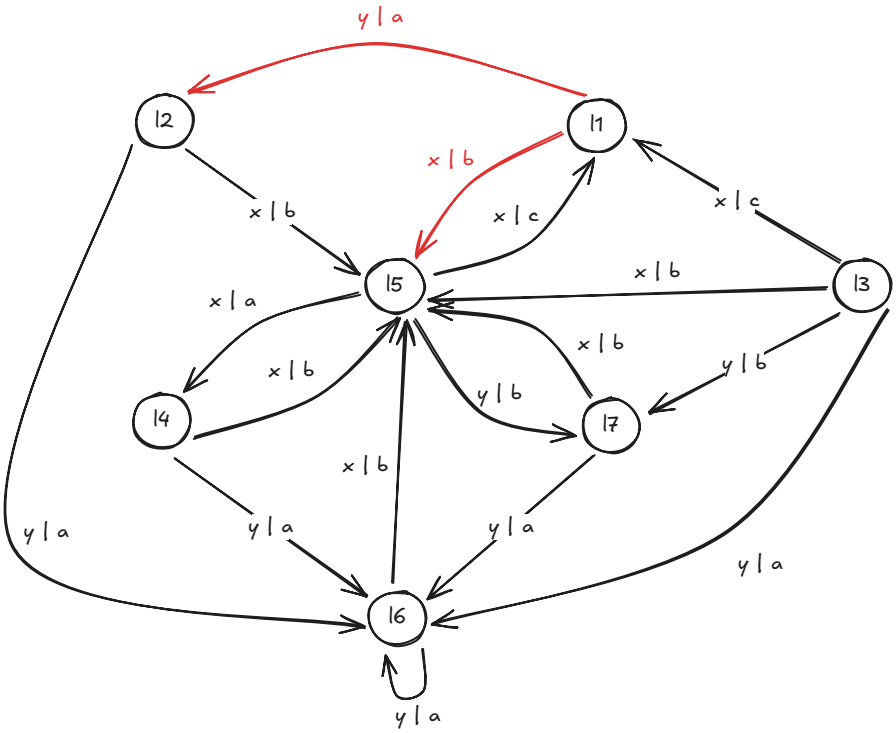


Figure 51 — FSM representation of the decision tree model from Figure 50.

However, not all transitions in the FSM representation are deterministic. For example, from State l_5, there are two possible transitions for the same input x, reflecting ambiguity in the model.

This example demonstrates how the structure of the decision tree translates into the FSM representation.

The FSM representation is a thought experiment to analyze the temporal progression of feature vectors over time. The FSM serves as an auxiliary tool for conceptual clarity, which motivates and illustrates a test coverage metric for decision tree models in the following. The primary model and object of study throughout this section remains the decision tree.

5.4.2 Leaf Coverage Criterion

The FSM representation of the decision tree model motivates the development of coverage metrics for decision trees, inspired by established coverage metrics for FSMs. We introduce *leaf coverage* as a coverage metric for decision tree models equivalent to state coverage in FSMs. As decision trees inherently compress system behavior by grouping redundant feature vectors into the same leaf node, leaf coverage effectively measures the extent to which the learned behavior has been exercised during testing.

We define leaf coverage as the ratio of the number of leaves reached during testing to the total number of leaves in the decision tree:

$$g_{\text{cover}} = \frac{|R_L|}{|V_L|}, \tag{68}$$

where R_L is the set of leaves reached during testing and V_L is the set of all leaves in the decision tree. By design, the leaf coverage metric corresponds to state coverage in the FSM representation of the decision tree.

5.4.3 Automatic Test Generation

The objective of testing is to identify a sequence of inputs that explores as much of the relevant system behavior as possible, starting from the current state of the system. A secondary goal is to minimize the number of steps required to achieve maximum behavioral coverage. In the following, we introduce a heuristic approach for generating input traces that considers these goals with respect to leaf coverage in the decision tree model.

The identification of an optimal input trace to maximize the leaf coverage on the decision tree model is equivalent to finding a minimal Hamilton path on the FSM representation of the decision tree model. The Hamilton path problem is known to be NP-complete making the computational effort high [152]. Thus, we propose a heuristic algorithm for automatic test generation for a system $\mathcal{S}$ based on the decision tree model $\mathcal{T}$. Our approach iteratively visits leaves in the decision tree, heuristically deciding on the next leaf to visit.

The algorithm uses the following helper functions:

- getLeaf($\boldsymbol{f}$): Returns the leaf that is reached with the feature vector $\boldsymbol{f}$.
- generateVectors(n): Gives a set of all possible feature vectors, i.e., all combinations of the input-output alphabet of length n.
- getInputTrace($\boldsymbol{f}, l$): Returns the input trace needed to reach a target leaf l from the leaf that is reached with the feature vector $\boldsymbol{f}$. This problem corresponds to the identification of a minimal path from one node to another node in the FSM representation of the decision tree model. Dijkstra's algorithm (minimal path) or the algorithm by Floyd-Warshall (minimal spanning tree) solve this problem [153].
- getReachableLeaves($\boldsymbol{f}$): Returns the set of leaves that are reachable from the feature vector $\boldsymbol{f}$. This set is determined from a minimal spanning tree of the FSM representation of the decision tree model.

Further, there are two functions that determine the heuristic to select a next leaf and influence the interaction with the system during learning, respectively.

- selectNextLeaf($V_{\text{next}}, R_L, \mathcal{T}, \boldsymbol{f}$): Selects a next leaf from the reachable leaves V_{next} of the current feature vector $\boldsymbol{f}$. Listing 4 provides pseudocode for this function, which implements two different heuristics. The variable $s_{\text{heuristic}}$ determines whether the 'length-based' heuristic or the 'coverage-based' heuristic is used. In the length-based heuristic, we use the next leaf reachable in a minimum number

```
1:  function selectNextLeaf(V_next, R_L, T, S, f, s_heuristic)
2:      ▷ length-based
3:      if s_heuristic then
4:          v ← argmin_{v∈V_next} |getInputTrace(f, v)|
5:      end
6:      ▷ coverage-based
7:      else
8:          M_next ← {}
9:          for v ∈ V_next do
10:             i ← getInputTrace(f, l)
11:             f_next ← getNewHistory(f, i, T, S)
12:             M_next ← M_next ∪ {⟨v, f_next⟩}
13:         end
14:         v ← argmax_{v∈V_next} |getReachableLeaves(M_next[v]) \ R_L|
15:     end
16:     return v
17: end
```

Listing 4 — Function to select the next leaf to visit. The input is the set V_{next} of reachable leaves, the set R_L of reached leaves, the decision tree $\mathcal{T}$, the system $\mathcal{S}$, the current feature vector $\boldsymbol{f}$, and the parameter $s_{\text{heuristic}}$ to select the heuristic.

of steps in Line 3 of Listing 4. We use the function getInputTrace($\boldsymbol{f}, v$) to get the shortest path to reach the leaf v with the current feature vector $\boldsymbol{f}$.

In the coverage-based heuristic, we determine in Line 12 how many unreached leaves are reachable for a potential next leaf. For this, we find the feature vector observed for every next leaf in the for-loop from Line 7 to Line 11.

- getNewHistory($\boldsymbol{f}, \boldsymbol{i}$): Returns the new history, i.e., new feature vector after an input trace $\boldsymbol{i}$ is applied. The algorithm is given in Listing 5. The algorithm consecutively updates the feature vector as defined in Equation 65.

  There are two options for an update. In case that the system can be accessed interactively, the new output is directly observed on the system by calling getOutput($\mathcal{S}, \boldsymbol{f}$) in Line 4. Otherwise, the prediction of the decision tree model is used in Line 7. By this, the new feature vector after the application of the input sequence $\boldsymbol{i}$ to the system is created based on the prediction of the decision tree model. The new feature vector is returned in Line 11.

Our approach to automated test generation is defined in Listing 6. We observe the system and initialize the list R_L of reached leaves with the leaf reached with the observed feature vector $\boldsymbol{f}$ using the function getLeaf() in Line 2. The set $\mathcal{O}_{\text{test}}$ of generated test inputs is initialized with the input trace of the current observation. A map M is initialized and then filled with tuples of feature vectors and the set of

```
1:  function getNewHistory(f, i, T, S)
2:      for i ∈ i do
3:          if S is interactive then
4:              o ← getOutput(S, f)
5:          end
6:          else
7:              o ← d_T(f)
8:          end
9:          f ≪ ⟨o, i⟩
10:     end
11:     return f
12: end
```

Listing 5 — Function to determine the feature vector after the application of the input trace $\boldsymbol{i}$. This is done by a consecutive update of the feature vector $\boldsymbol{f}$ querying either the decision tree model $\mathcal{T}$ or the system $\mathcal{S}$.

```
1:  function generateTests(S, T, s_heuristic)
2:      f ← observe(S)
3:      R_L ← {getLeaf(f)}
4:      i, o ← f
5:      O_test ← {i}
6:      M ← {}
7:      for f ∈ generateVectors(n) do
8:          m ← getReachableLeaves(f)
9:          M ← M ∪ {⟨f, m⟩}
10:     end
11:     V_next ← M[f] \ R_L
12:     while V_next ≠ ∅ do
13:         v ← selectNextLeaf(V_next, R_L, T, S, f, s_heuristic)
14:         i ← getInputTrace(f, l)
15:         O_test ← O_test ∪ {i}
16:         R_L ← R_L ∪ {v}
17:         f ← getNewHistory(f, i, T, S)
18:         V_next ← M[f] \ R_L
19:     end
20:     return O_test
21: end
```

Listing 6 — Algorithm for automatic test generation with a decision tree model $\mathcal{T}$ for the system $\mathcal{S}$ using the heuristic defined by $s_{\text{heuristic}}$. Reachable leaves that are not yet visited are identified to optimize the leaf coverage metric.

reachable leaves in Line 7 to 10. The leaves are determined by the reachableLeaves($\boldsymbol{f}$) function.

Afterward, the algorithm generates test inputs by consecutively visiting the next selected leaf in Line 12 to 19 of Listing 6. A set of reachable candidate leaves V_{next} is initialized in Line 11. The function selectNextLeaf() from Listing 4, determines the leaf that is visited in the next iteration. Line 15 to 18 update the set of generated test inputs $\mathcal{O}_{\text{test}}$ with the input trace to reach the selected leaf and add the leaf to the set of reached leaves R_L. Further, the feature vector is updated using the getNewHistory() function in Line 17 and the set of next candidate leaves V_{next} is updated in Line 18. The algorithm continues until all reachable leaves are visited, i.e., the set V_{next} is empty. Finally, the set of generated test inputs $\mathcal{O}_{\text{test}}$ is returned in Line 20.

S. Plambeck et al. [33] shows that the algorithm in Listing 6 reaches full leaf coverage for a decision tree model with a *strongly connected* FSM representation, i.e., an FSM representation, where every leaf is reachable from every other leaf.

Theorem 5.5: For a decision tree model with a strongly connected FSM representation as follows

$$\forall \boldsymbol{f_1}, \boldsymbol{f_2} \in \mathbb{F} : \exists \boldsymbol{i} \in I^* \text{ such that } \boldsymbol{f_2} = \text{getNewHistory}(\boldsymbol{f_1}, \boldsymbol{i}, \mathcal{T}),$$

The algorithm from Listing 6 always reaches full leaf coverage, i.e., $g_{\text{cover}} = 1$.

Proof: We assume a decision tree model that has a strongly connected FSM representation as defined above. Thus, for every feature vector $\boldsymbol{f}$, we have:

$$M[\boldsymbol{f}] = V_L, \tag{69}$$

for the map M of Listing 6. We prove the above theorem by contradiction. We assume that the algorithm does not reach full leaf coverage, i.e.,

$$V_L \setminus R_L \neq \emptyset. \tag{70}$$

Without loss of generality, we assume that the last feature vector used in Listing 6 is $\boldsymbol{f}$. As the algorithm terminated, we know that

$$M[\boldsymbol{f}] \setminus R_L = \emptyset. \tag{71}$$

This is a contradiction to the initial assumptions of Equation 69 and Equation 70. Thus, Theorem 5.5 holds. □

If the FSM representation is not strongly connected, i.e., not all leaves are reachable from every other leaf, Listing 6 might not reach all leaves. An example is a system with a dead-end state such as an error state, which does not allow returning to nominal behavior. In this case, the choice of heuristic for the selection of a next leaf in Line 13 of Listing 6 influences the performance of the algorithm. For the length-based heuristic, a dead-end might be reached early if it is the closest reachable leaf. The coverage-based heuristic instead, maximizes the number of leaves that are reachable in the next step.

5.4.4 Case Study

Setup
Examples: Coffee machine example from Section 3.7.1
Scenarios: Testing
Learning Set: Complete learning set $\mathcal{O}_n^O$ for $n = 2$
Metrics: Leaf coverage g_{cover}, i.e., the number of leaves reached by the test input trace

We present a case-study for test case generation with a decision tree model for the coffee machine example. We consider the Mealy machine model of the coffee machine from Section 3.7.1. The system has the four input symbols BUTTON, CLEAN, POD, and WATER and the three output symbols ok, error, and coffee.

Figure 52 shows a learned decision tree model for $n = 2$. The figure uses the abbreviations B: BUTTON, C: CLEAN, P: POD, W: WATER for the inputs. With $n = 2$, the decision tree considers three features, i.e., $\boldsymbol{f} = [f_1, f_2, f_3]$. The feature f_1 is the observation at the second last timestep, f_2 is the observation at the previous timestep, and f_3 is the current input. For brevity, the figure labels only the left branch of every node. The right branch is enabled for all other possible feature values. The decision tree model of this example does not form an exact representation of the system, because the system has ambiguous observations at the self loops with the label WATER-ok at States 3 and 4. Leaf l_3 is an impure leaf because of the ambiguous observations and might predict wrong outputs. The decision path from root node to Leaf l_3 indicates that only a pod or only water would be sufficient to produce a coffee, which is obviously not the case.

We use our decision tree model to generate a test input sequence with the length-based heuristic and no interaction with the system:

1) At the start of testing, we assume our system to be in State 2 of the FSM in Figure 11 with a current history [C-ok , P-ok , W]. Thus, the first leaf reached is l_4.
2) The output corresponding to l_4 is ok. With the current history, we are able to reach all other leaves in the future. The input B allows reaching a next unreached leaf with a distance of 1. Thus, B is chosen as the next input and Leaf l_3 is reached with a current history [P-ok , W-ok , B].
3) The output corresponding to l_3 is coffee. Again, all other leaves are reachable with this history. The input C allows reaching a next unreached leaf with a distance of 1. Thus, C is chosen as the next input and Leaf l_5 is reached with a current history [W-ok , B-coffee , C].
4) The output corresponding to l_5 is ok. With the input trace [W , B] the unreached leaf l_1 is visited in a closest distance. Thus, [W , B] is chosen as the next input and results in the history [C-ok , W-ok , B].

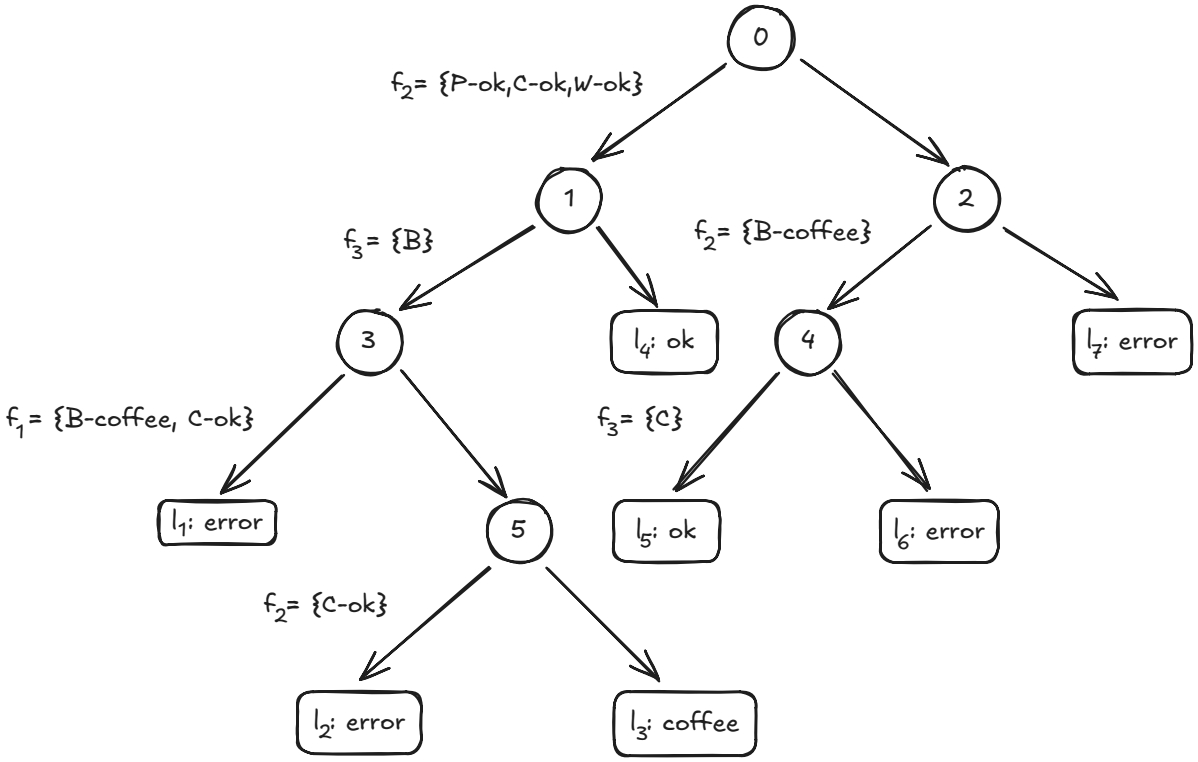


Figure 52 — Decision tree model of the coffee machine from Figure 11 with $n = 2$.

5) The output corresponding to l_1 is **error**. From Leaf l_1, Leaf l_7 can be visited in 1 step using any input. We choose the input **P**.
6) No other unreached leaf is reachable from this state and, thus, the algorithm terminates.

The resulting test input trace [**W , B , C , W , B , P**] reaches a leaf coverage of $\frac{5}{7}$. Applying the same input trace on the original automaton representation in Figure 11, we observe that full state coverage on the FSM is reached for this example.

The example shows the practical applicability of our decision tree model in generating a test input sequence that achieves high behavioral coverage. Our test case generation approach leverages the decision tree structure to explore the input space and identify relevant test cases. We introduce a new coverage metric inspired by coverage metrics from FSMs. By this, we are able to use the practical advantages in the learning requirements of decision tree models, such as the learning from non-deterministic behavior and bounded history, and still maintain a tool for test generation.

5.5 Summary

The results of this chapter answer the research question **Q2 — What is the performance of decision trees that integrate temporal information for modeling of CPSs?** through the five interconnected sub-questions identified at the beginning of this chapter.

Q2.1 — How to integrate temporal information into decision trees for modeling CPSs? We develop a novel preprocessing methodology that transforms temporal observations into feature vectors suitable for DTL by incorporating bounded history information. Our approach maps observed traces of the system to discrete observations of bounded history, creating two complementary variants: a language-based mapping that determines trace validity and an output-based mapping that predicts future outputs. This temporal integration addresses the fundamental challenge of representing sequential dependencies in CPS behavior. The preprocessing enables learning from non-reset observations, providing a practical advantage over automata learning, where system resets are often impossible.

Q2.2 — How do decision tree models with temporal information relate to FSMs? We establish formal theoretical connections between decision tree models and FSMs. Our analysis proves that decision tree models learned from bounded history generally over-approximate the system behavior, resulting in languages that are supersets of the original Mealy machine languages. We show that this over-approximation stems from ambiguous observations caused by bounded history limitations — a fundamental constraint of learning from finite traces rather than a specific limitation of decision trees. The construction of FSM representations from decision tree leaf transitions provides a bridge between decision tree models and automata models.

Q2.3 — What is the quantitative approximation of decision tree models compared to the original system? We derive formal bounds on the prediction error for decision tree models and provide empirical validation of these theoretical results. The number of impure leaves in a decision tree model directly quantifies the presence and extent of ambiguities in observed system behavior, offering interpretable uncertainty indicators. Our error bounds establish that the maximum prediction error is bounded by twice the maximum distance between consecutive outputs in the discrete space, with tighter practical bounds determined by the specific decision tree structure. Empirical evaluations confirm high prediction accuracy in practice, with decision trees achieving performance comparable to neural networks while providing superior interpretability.

Q2.4 — How do regression trees contribute to the modeling of systems with continuous behavior? We extend our methodology to regression trees, enabling direct modeling of systems with continuous variables without explicit discretization. Regression trees automatically identify a discrete partitioning of the input space while maintaining the temporal integration capabilities of discrete decision trees. Thus, this approach is at the transition from discrete decision tree models to fully continuous

representations by accepting continuous features and producing piecewise constant approximations of the continuous output signal.

Q2.5 — How do decision tree models support applications such as test case generation? We develop novel test case generation strategies that exploit the decision tree structure through specialized coverage criteria. By constructing FSM representations from leaf transitions, we enable systematic test case generation guided by an innovative leaf coverage metric. The method successfully achieves comprehensive coverage while providing practical test case generation without requiring system resets.

Overall Assessment: Decision tree models represent a practical alternative to automata for learning discrete models of CPSs from observations. They successfully address the core limitations of deterministic automata while maintaining interpretability: handling non-deterministic systems, operating without system resets, and providing explicit uncertainty indicators through impure leaves. However, the restriction to discrete systems highlights the need for more expressive modeling approaches, directly motivating hybrid automata learning explored in the subsequent chapter.

These findings establish decision trees as a powerful intermediate step in our modeling hierarchy, bridging the gap between purely deterministic automata and the hybrid approaches necessary for capturing the full complexity of CPS behavior.

Chapter 6

Identification of Hybrid Automata

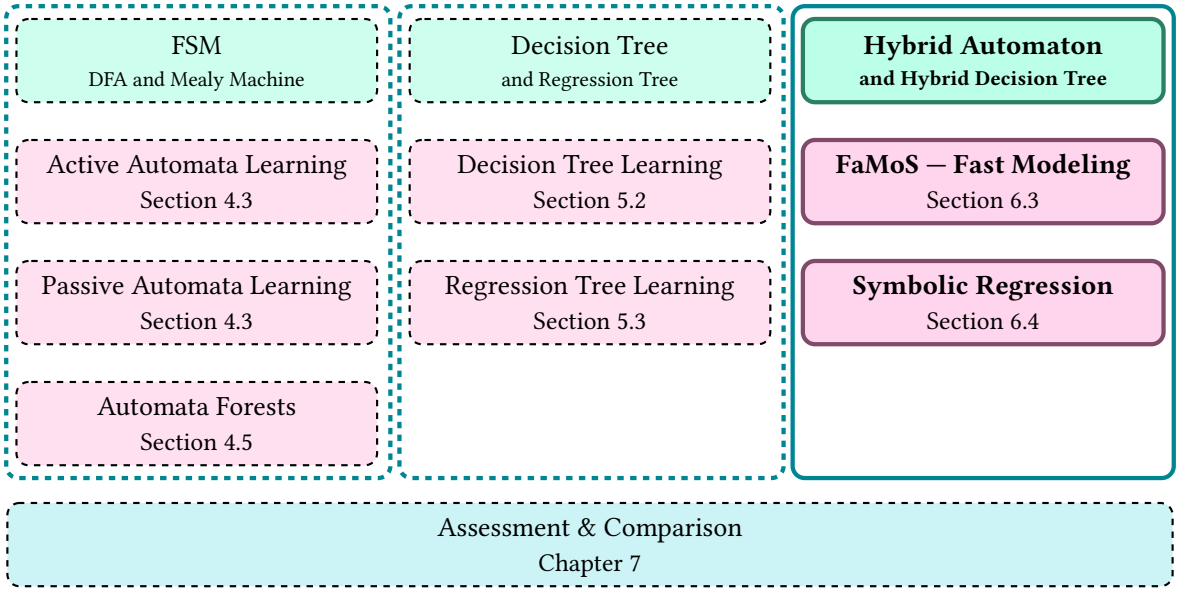


Figure 53 — Overview of the presented models (green) and learning approaches (pink) per model type in this work. This chapter presents approaches for learning hybrid automata models.

According to our view of CPSs, they exhibit both discrete and continuous dynamics. Thus, discrete models like DFAs or decision tree models cover only the discrete part of the system behavior or address the continuous aspects only partially, e.g., via regression trees. As established in our progressive modeling hierarchy of Figure 53,

the limitations of purely discrete approaches necessitate models that capture the interplay between discrete transitions and continuous evolution while maintaining interpretability. In this chapter, we explore hybrid automata as the final stage of our modeling progression — representing the most expressive approach capable of capturing both discrete and continuous dynamics of CPSs.

Figure 53 provides an overview of hybrid automata in the context of the methods presented in this thesis. Hybrid automata as introduced in Section 3.3 are a natural framework to model both the continuous and discrete aspects of a system.

Building on the insights from previous chapters, where purely discrete models revealed their limitations in capturing continuous dynamics, this chapter addresses the identification of hybrid automata. Our investigation answers research question **Q3 — How to learn a model to capture hybrid dynamics of CPSs?** by addressing the following interconnected questions:

- Q3.1 — Which subsequent steps are required for the identification of hybrid automata?
- Q3.2 — How to utilize decision trees for modeling of hybrid dynamics?
- Q3.3 — How can lightweight and intuitive algorithms enable efficient identification of hybrid automata?
- Q3.4 — How to migrate to a dynamics-based identification using symbolic regression?

We start by introducing the structure and concept of hybrid decision trees — our extension of traditional decision trees to capture hybrid dynamics. This bridges the gap between the discrete decision tree models from the previous chapter and the continuous dynamics requirements of CPSs. Our hybrid decision tree framework aligns the approaches presented by S. Plambeck et al. [34] [36], providing a unified foundation for both algorithmic variants, we propose. We discuss general methodological considerations for data-driven identification of hybrid automata, including evaluation strategies and metrics that account for both discrete and continuous aspects of the learned models and enable the comparison of different approaches.

Subsequently, we present the two parallel algorithms for hybrid automata identification in detail, each addressing different practical requirements. The first approach, FaMoS (Fast Modeling of Cyber-Physical Systems) [34], focuses on computational efficiency through lightweight procedures, making it suitable for real-time applications and large datasets. The second approach introduces a paradigm shift by integrating symbolic regression into the learning process, enabling dynamics-based identification utilizes identified dynamics from the beginning of the learning process [35], [36].

Both algorithms incorporate hybrid decision trees while leveraging distinct techniques to model continuous dynamics, demonstrating the versatility of our framework. Through an individual evaluation of both examples on extended scenarios that combine and extend examples from S. Plambeck et al. [34] [36], we establish their respective strengths and application domains.

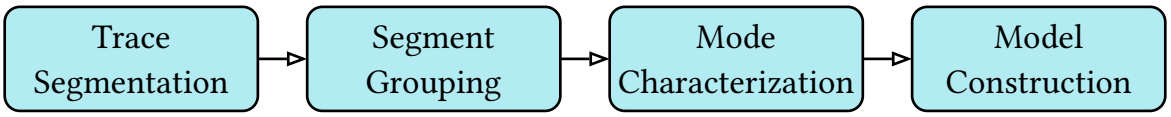


Figure 54 — Learning process for hybrid automata models consisting of four steps.

6.1 Related Work

Existing approaches for the identification of hybrid automata follow a similar learning procedure consisting of four steps as shown on Figure 54. The first step involves the segmentation of observed traces to separate distinct operating modes. Next, segment grouping is performed to cluster segments of similar dynamics. The third step is mode characterization, where the identified modes are analyzed and described by continuous dynamics. The final step is the construction of the model, where the identified modes and their dynamics are integrated into a cohesive hybrid automaton representation. There is a diverse set of methods for the individual steps, which are then combined in different ways. An overview of related approaches, addressing the learning steps of Figure 54, is given by S. Plambeck et al. [34].

O. Niggemann et al. [73] introduce a learning algorithm called HyBUTLA for learning hybrid timed automata. HyBUTLA uses wavelet analysis, i.e., similarities in the frequency domain of signals, to identify the transition points in the observed traces. Segment grouping is done together with the model construction using a PTA. Mode characterization, in the last step of the learning process, is done with recurrent neural networks. Unlike HyBUTLA, which uses neural networks for mode characterization, our approaches employ interpretable models such as symbolic expressions and decision trees.

Another approach for the identification of hybrid automata leverages dynamic grouping [24], [154]. The dynamic grouping combines the trace segmentation and segment grouping steps. For the mode characterization, support vector machines and random forests are used. The final hybrid automaton is represented with a heterogeneous petri net.

X. Yang et al. [120] identify transition points through extrema in the derivatives of signals. The grouping by X. Yang et al. [120] is done with linear matrix inequalities assuming that the dynamics are represented by matrix differential equations. Consequently, mode characterization is done by identifying the matrices of the differential equations. The hybrid automaton is finally constructed via a PTA. We built on this approach in our first algorithm to improve efficiency and speed of the learning process and exchange the construction of a hybrid automaton with a hybrid decision tree.

S. Gaucel et al. [155] also learns dynamics of matrix ordinary differential equations, but only for a single system mode. The approach uses symbolic regression to identify the flow functions of the system. D. L. Ly et al. [156] present a clustering approach using symbolic regression that implements the segmentation, grouping, and mode

characterization in a joined fashion. The second approach presented in this chapter uses symbolic regression as well and focuses on a dynamics-driven identification of hybrid automata.

N. Kochdumper et al. [157] use k-means clustering on a prediction error metric, which directly groups data points, thus, skipping the segmentation step. The modes are represented by polynomials in the system variables. N. Kochdumper et al. [157] use decision trees to construct a hybrid automaton in the last step. Our approaches use decision trees extended with flow function as the final hybrid model of the system.

In summary, existing approaches differ in their strategies for segmentation, grouping, and mode characterization. Our methods build on these foundations by emphasizing interpretability and efficiency, and by introducing hybrid decision trees and symbolic regression for model identification.

6.2 Hybrid Decision Tree — Structure, Modeling Process & Evaluation

As regression trees in Chapter 5, the identification of hybrid automata allows learning models from continuous data. Decision trees as introduced in Section 3.1 are a common method for classification and regression tasks. We extend the decision tree model to hybrid decision trees, which incorporate discrete transitions and continuous dynamics of hybrid automata.

Definition 6.1: A *Hybrid Decision Tree* is a 5-tuple $\mathcal{T} = (X, V_H, E_H, Q, F)$ where

- $X = \{x_0, x_1, x_2, ..., x_n\} = I \cup O \cup S$ is the set of system variables, which consist of input variables I, output variables O, and state variables S. The state variables may include the continuous time t. Also, derivatives of variables may form individual variables in X.
- V_H, E_H are a set of nodes and edges, respectively, which form a tree, i.e., a connected, acyclic graph,
- Q is a set of system modes, and
- F is the set of flow functions. A flow function $f_q \in F$ defines the change of state variables S as well as the current output variables O within the mode $q \in Q$.

The tree V_H, E_H represents a classification function $d_H : Q \times X \to Q$, which encodes the transitions between modes of the system based on the current state and system variables.

The system variables, modes, and flow functions of the hybrid decision tree are equivalent to those of the hybrid automata from Definition 3.6. The transition guards K and the discrete transitions T of the hybrid automaton are represented by the tree structure of the hybrid decision tree, i.e., the edges E_H and nodes V_H and the classification function d_H.

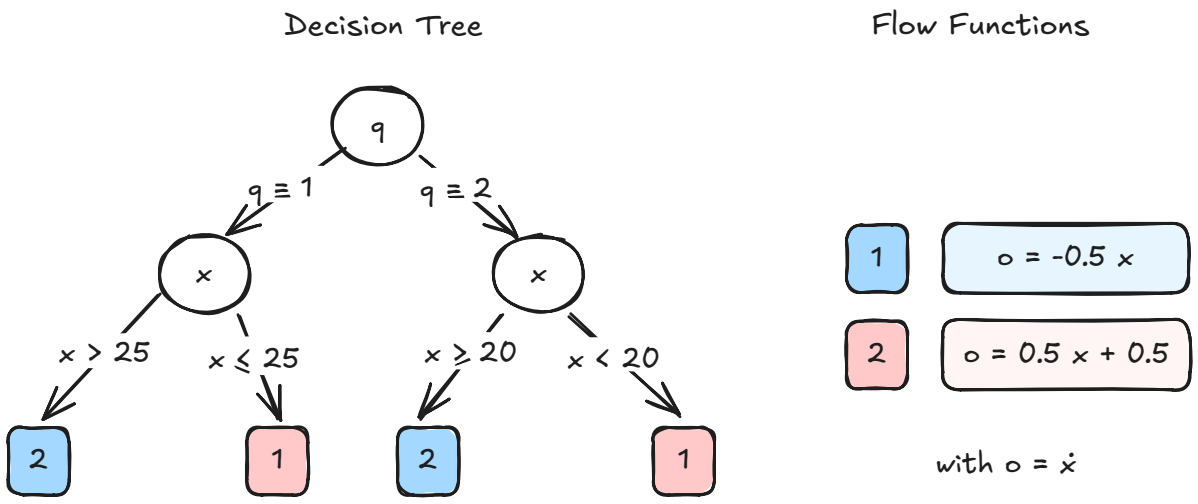


Figure 55 — Hybrid decision tree consisting of a decision tree (left) and flow functions (right). The modes are indicated by colors.

Example 11: Figure 55 gives an example of a hybrid decision tree for the hybrid automaton of the boiler example from Figure 22 with the variables $X = \{x, \dot{x}\}$. Every inner node of the decision tree splits on one of the variables or the current mode q of the system. The labels of the edges of the decision tree are the splitting rules. For example, the root node in Figure 55 decides on the mode q of the system. If the system is in Mode 1, i.e., heating mode, the left child node is selected. If the system is in Mode 2, i.e., cooling mode, the right child node is selected. The leaf nodes classify one of the modes $q \in \{1, 2\}$. Finally, the flow functions map these modes to descriptions of the continuous dynamics as shown on the right of Figure 55.

For all systems considered in this work, the output variables are a subset of the state variables. We learn flow functions for the output variables based on the state variables of the system. The formalization holds for more general cases, while specific implementations may require additional considerations to extend to multi-output scenarios.

The input for the learning of hybrid automata is a set of traces, i.e., time series of variables observed from the system. As introduced Figure 54 shows the learning process for hybrid automata [34]. In the following, we analyze and discuss the four learning steps and their evaluation:

1) The first step segments the traces by identifying transition points, which mark changes in the continuous dynamics of the system. We evaluate the segmentation step by comparing the identified transition points to ground truth transition points.
2) The second step groups the segments into distinct dynamic modes.
3) The third step learns the flow functions for each mode. We evaluate the grouping and the characterization steps by the accuracy of the found flow functions per group.

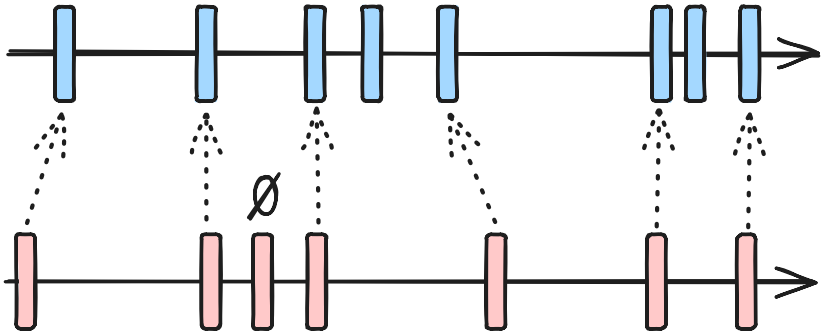


Figure 56 — Illustration of the mapping w for the segmentation objective Ω_{seg}. The ground truth transition points are indicated in red, while the found transition points are blue. The mapping for this example is $w = \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,\emptyset\rangle, \langle 4,3\rangle, \langle 5,5\rangle, \langle 6,6\rangle, \langle 7,8\rangle\}$.

4) The fourth step constructs the overall hybrid decision tree model. We evaluate the model by comparing its predicted trace to the ground truth behavior of the system.

Step 1) segments the trace by identifying transition points $\boldsymbol{p}$. The transition points are identified by analyzing the continuous dynamics of the system and detecting changes in the behavior of the observed variables. The transition points are used in the segmentation step to split the traces into segments. We evaluate the segmentation step by comparing the identified transition points to ground truth transition points of evaluation traces using the objective function Ω_{seg}. The objective function uses a mapping w between ground truth transition points and their closest transition point found in the segmentation step. If the number of found transition points is less than the number of ground truth transition points, w may contain $\emptyset$ if the closest found transition point is mapped to another ground truth transition point that is closer. An example of such a mapping is illustrated by Figure 56. Missed or additionally found transition points are penalized as an additive term with the factor c_O in the objective function as follows

$$\Omega_{\text{seg}}(\boldsymbol{p}_{gt}, \boldsymbol{p}_f, w) = \frac{1}{|\boldsymbol{p}_f|}\left(c_O \cdot \big||\boldsymbol{p}_f| - |\boldsymbol{p}_{gt}|\big| + \sum_{\langle i,j\rangle \in w, j\neq\emptyset} |\boldsymbol{p}_{gt}(i) - \boldsymbol{p}_f(j)|\right), \quad (72)$$

where $\boldsymbol{p}_{gt}$ are the ground truth transition points, and $\boldsymbol{p}_f$ are the transition points that are found in the trace segmentation step.

In Step 2) of the learning process, the segments are grouped so that each group corresponds to a distinct dynamic mode of the hybrid automaton. This grouping is, e.g., based on the similarity of the segment behavior and their corresponding flow functions or the accuracy of the flow functions.

Step 3) of Figure 54 characterizes the behavior of each mode from the data by identifying the flow functions. The flow functions describe how the system evolves in each mode and are typically represented by differential equations or other mathematical

```
1: function inferHybridDT(x, d_H, F)
2:     q ← d_H(x)
3:     f ← F[q]
4:     o ← f(x)
5:     return o
6: end
```

Listing 7 — Inference of the hybrid decision tree. The relevant flow function is selected using the decision tree and then evaluated for the sample $\boldsymbol{x}$.

expressions. For every group of segments identified in the previous step, we learn a flow function that describes the continuous dynamics of the system in that mode.

The evaluation of the grouping and characterization steps compares the found groups and flow functions to ground truth modes of the system using the objective function Ω_{group}. The objective function uses the error of the learned flow functions per group weighted by their size $|g|$, i.e., the number of samples in the group. Further, we penalize a divergence in the number of ground truth groups and the groups found in the grouping step by a multiplicative factor as follows

$$\Omega_{\text{group}}(G_{gt}, G_f) = \frac{1 + \big||G_{gt}| - |G_f|\big|}{\sum_{g \in G_f} |g|} \cdot \sum_{g \in G_f} e_g \cdot |g|, \tag{73}$$

where G_{gt}, G_f are the ground truth and found groups, respectively. The error e_g of the group g is, e.g., the mean squared error of the segments in the group to the segments of the learning set. Further, we evaluate the flow functions by comparing the identified flow functions to ground truth flow functions.

In Step 4) of the learning process, the complete model is constructed by combining the identified modes and their flow functions as well as defining the transitions between modes. The approaches presented in this work construct a hybrid decision tree, which captures the relationships between the modes and their flow functions. The final model is evaluated on a test set of traces by comparing the predicted behavior of the model to the actual behavior of the system and assessing the MSE between the predicted and actual outputs.

For prediction, we apply the decision function d_{H} of the hybrid decision tree on the samples of the test set as given in Listing 7. Given a sample $\boldsymbol{x}$, the decision function first determines the mode q of the sample by evaluating the decision tree. Using this mode q, we identify the correct flow function. This function is used to evaluate the system dynamics using the variable values in $\boldsymbol{x}$.

We now turn to concrete implementations of methods for the identification of hybrid decision trees.

6.3 Fast Model Learning for Hybrid Systems

In this section, we introduce FaMoS, a fast and efficient algorithm for learning hybrid decision trees from observed traces, where every trace $\boldsymbol{s}$ consists of sub-traces of inputs and outputs, i.e., $\boldsymbol{s} = [\boldsymbol{i_1}, ..., \boldsymbol{i_k}, \boldsymbol{o_1}, ..., \boldsymbol{o_n}]^T$ as introduced in Section 3.4. In the following, we introduce the underlying model structure, and the learning steps segmentation, grouping, mode characterization, and model construction. FaMoS is originally presented by S. Plambeck et al. [34].

The approach is inspired by X. Yang et al. [120] and aims for a fast and efficient learning of hybrid decision trees. We consider time-discrete systems that depend on both, past state variables (autoregressive) and on current input variables (exogeneous). Such models are referred to as AutoRegressive eXogenous (ARX) models [6].

Every mode of the hybrid decision tree follows an ARX model and is represented by a time-discrete Linear Time-Invariant (LTI) model using a difference equation. The state equation of a difference equation is defined as follows

$$\boldsymbol{y}(t+1) = \boldsymbol{A}\boldsymbol{y}(t) + \boldsymbol{B}\boldsymbol{u}(t). \tag{74}$$

The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are system matrix and input matrix, respectively, while the vectors $\boldsymbol{y}$ and $\boldsymbol{u}$ are state and input vector, respectively. The input vector is constructed from the current input samples, i.e.,

$$\boldsymbol{u}(t) = [\boldsymbol{i_1}[t], \cdots, \boldsymbol{i_k}[t]]^T. \tag{75}$$

The state vector consists of current and past output variables, i.e.,

$$\boldsymbol{y}(t) = [\boldsymbol{o_1}[t], \cdots, \boldsymbol{o_n}[t], \cdots, \boldsymbol{o_1}[t-\kappa], \cdots, \boldsymbol{o_n}[t-\kappa]]^T. \tag{76}$$

Thus, Equation 74 forms a difference equation of the order $\kappa + 1$ [158].

We use the model structure of Equation 74 to represent the flow functions F of the hybrid automaton. ARX models are particularly well-suited for this purpose, as they are able to capture the dynamics of a wide range of real-world systems while remaining interpretable and computationally efficient. By adopting this model type, FaMoS enables the learning of hybrid decision trees that are both transparent and practical for analysis and control, and balance expressiveness with efficiency. The learning procedure of FaMoS follows the steps of Figure 54 for which we describe the details in the following.

6.3.1 Trace Segmentation

The trace segmentation identifies a change in the dynamics of the system behavior by detecting transition points in the observed traces. Transition points are determined for the output traces $\boldsymbol{o_1}, ..., \boldsymbol{o_n}$ individually. In FaMoS, the trace segmentation is realized with a sliding window approach similar to the approach of C. Truong et al. [159].

For every sample of a trace, windows of immediate past and immediate future are compared. A large deviation between the two windows indicates a change in the system behavior. The Euclidean distance quantifies the deviation between the imme-

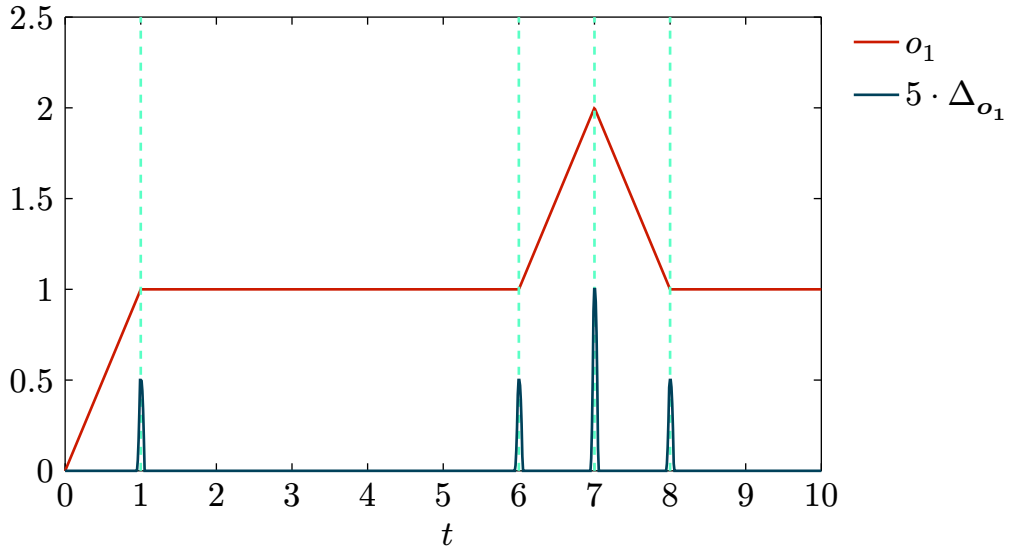


Figure 57 — Example of the trace segmentation step in FaMoS using $\omega = 5$ and a sampling rate of $1/100$. The scaled deviation function is shown in blue. The detected transition points are shown as vertical dashed lines.

diate past and the immediate future. Thus, on an output trace $\boldsymbol{o_i}$, the deviation between immediate future and immediate past for a time point t is determined as follows

$$\begin{aligned}\Delta_{\boldsymbol{o_i}}(t) = \sum_{j=0}^{\omega-1} |(\boldsymbol{o_i}[t-\omega+j] - \boldsymbol{o_i}[t-\omega]) \\ -(\boldsymbol{o_i}[t+1+j] - \boldsymbol{o_i}[t+1])|,\end{aligned} \tag{77}$$

where ω gives the window width and is a parameter of the trace segmentation. The subtraction of $\boldsymbol{o_i}[t-\omega]$ and $\boldsymbol{o_i}[t+1]$, respectively, aligns both windows to an initial value of 0. Evaluating the deviation for all time points $t \in [\omega, l-\omega]$, where l is the length of the trace, we have a deviation trace $\Delta_{\boldsymbol{o_i}}$. Established algorithms to detect local maxima (peak detection) are applied on the deviation trace to extract the transition points.

> *Example 12*: Figure 57 shows an example of the trace segmentation step in FaMoS. The blue line shows the deviation function for a trace of the variable o_1, shown in red. The transition points are detected as local maxima in the deviation function and are indicated by the vertical dashed lines.

The sliding-window trace segmentation is applied to the output traces individually. Additionally, derivatives up to an order κ_{seg} are considered to detect transition points in higher order dynamics. Finally, all transition points are collected. Transition points that are within a distance of ω_{size} are combined into a single transition point. Traces $\boldsymbol{s}$ are then split into segments at the transition points.

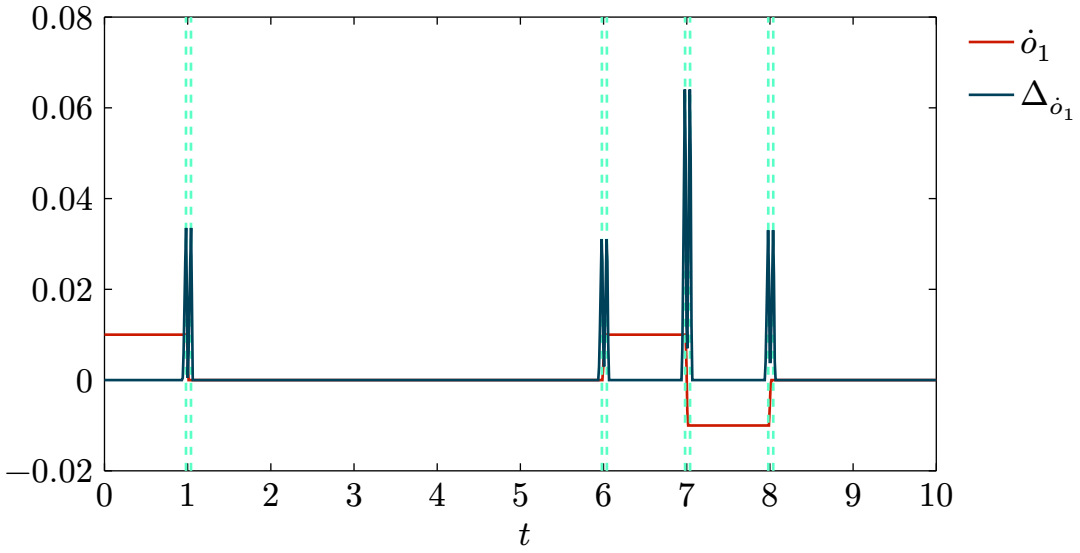


Figure 58 — Trace segmentation on the gradient of o_1 representing an example of a numerically calculated higher order derivative.

Example 13: Figure 58 shows the trace segmentation on the gradient Δo_1 of o_1, representing a numerically calculated derivative $\dot{o}_1$. The identified transition points are within a range of $\omega_{\text{size}} = 0.05$ of the ones from Example 12 and are combined into a single transition point. The resulting segments are

$$\begin{aligned} \Diamond \boldsymbol{o_1^1} &= \boldsymbol{s}[0,1], \\ \Diamond \boldsymbol{o_1^2} &= \boldsymbol{s}[1,6], \\ \Diamond \boldsymbol{o_1^3} &= \boldsymbol{s}[6,7], \\ \Diamond \boldsymbol{o_1^4} &= \boldsymbol{s}[7,8], \\ \Diamond \boldsymbol{o_1^5} &= \boldsymbol{s}[8,10]. \end{aligned} \tag{78}$$

6.3.2 Segment Grouping

Segments exhibiting similar behavior are grouped in the segment grouping step. Each group corresponds to a distinct dynamic behavior, i.e., a mode of the hybrid automaton. FaMoS groups segments with a DTW-inspired comparison, which is a fast and intuitive similarity check. The grouping process is illustrated in Listing 8. The process first identifies groups for each output individually and then combines these groups to global groups.

In Line 2, the derivative κ_g of the segments is determined, where κ_g is an input parameter for the grouping process. The choice of the derivative depends on the system dynamics. For hybrid automata whose behavior is defined by higher order dynamics the similarity in derivatives is usually relevant for the grouping as it does not depend, e.g., on the initial values of the state variables.

1: **function** groupSegments$\left(\Diamond \boldsymbol{s}_1, ..., \Diamond \boldsymbol{s}_K, \kappa_g, \text{useLMI}\right)$
2: $\quad \Diamond \boldsymbol{o}_1^{\kappa_g}, ..., \Diamond \boldsymbol{o}_K^{\kappa_g} \leftarrow$ createDerivative$(\Diamond \boldsymbol{s}_1, ..., \Diamond \boldsymbol{s}_K, \kappa_g)$
3: $\quad \text{Pairs} \leftarrow \left\{\langle \Diamond \boldsymbol{o}_i^{\kappa_g}, \Diamond \boldsymbol{o}_j^{\kappa_g} \rangle : \left\{ \Diamond \boldsymbol{o}_i^{\kappa_g}, \Diamond \boldsymbol{o}_j^{\kappa_g} \right\} \subseteq \left\{ \Diamond \boldsymbol{o}_1^{\kappa_g}, ..., \Diamond \boldsymbol{o}_K^{\kappa_g} \right\}\right\}$
4: $\quad d \leftarrow$ dimension$(\Diamond \boldsymbol{s}_1^{\kappa_g})$
5: $\quad$ **for** $j \in [1, ..., d]$ **do**
6: $\quad\quad$ **for** $\langle \Diamond \boldsymbol{o}_{j,x}^{\kappa_g}, \Diamond \boldsymbol{o}_{j,y}^{\kappa_g} \rangle \in$ Pairs **do**
7: $\quad\quad\quad \boldsymbol{S}_j[x, y] \leftarrow$ dtwBasedComparison$(\Diamond \boldsymbol{o}_{j,x}^{\kappa_g}, \Diamond \boldsymbol{o}_{j,y}^{\kappa_g})$
8: $\quad\quad$ **end**
9: $\quad\quad G_j \leftarrow$ thresholdGrouping$(\boldsymbol{S}_j)$
10: $\quad$ **end**
11: $\quad G \leftarrow$ globalGrouping$(G_1, ..., G_d)$
12: $\quad$ **if** useLMI **then**
13: $\quad\quad G \leftarrow$ lmiBasedRefinement(G)
14: $\quad$ **end**
15: $\quad$ **return** G
16: **end**

Listing 8 — Grouping process in FaMoS: first, the derivative κ_g, which is used for the grouping, is determined. Then, all possible pairs of segments are created. For every pair and every variable, DTW-inspired comparisons are executed. The results are stored in a comparison matrix $\boldsymbol{S}$ for each variable. Groups per variable are combined to global groups. An optional LMI-based refinement of the grouping is performed.

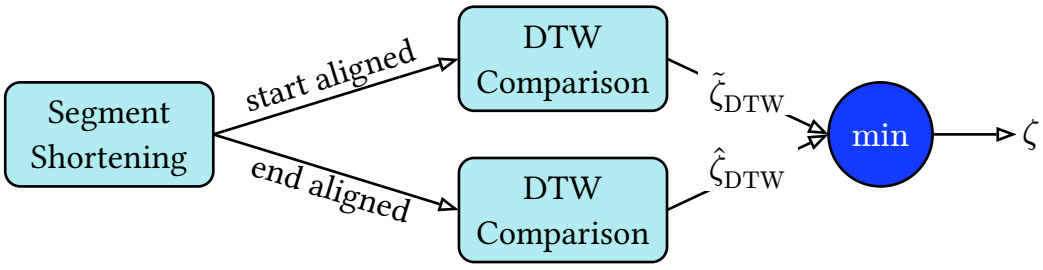


Figure 59 — Adaptation of DTW for FaMoS: the segments are either aligned at the start or at the end. The minimum of the two similarities is used as the final similarity.

A DTW-inspired comparison is performed for each pair $\langle \Diamond \boldsymbol{o}_{j,x}^{\kappa_g}, \Diamond \boldsymbol{o}_{j,y}^{\kappa_g} \rangle$, $j \in \{1, ..., d\}$ of segments, where d is the number of output signals as given in the

for-loops in Line 5 and 6. Section 3.5 introduces DTW as a technique to compare two traces [108]. Here, we use an adaption of DTW, which is illustrated in Figure 59.

The detected segments are often of different lengths, but share a fixed sampling frequency. To enable the comparison, we align the segments either at the start (truncating the end of the longer segment) or at the end (truncating the beginning of the longer segment), and apply DTW to both alignments. The minimum of the two resulting values is used as the final metric. A low DTW-metric indicates high similarity between segments. The DTW-metrics from all comparisons form a similarity matrix $\boldsymbol{S}_j$ for each output o_j in Line 7 of Listing 8, which quantifies the similarity between all segment pairs as described in Equation 7.

On the similarity matrix, groups are found by defining thresholds $\bar{\zeta}_i$ for every row i in the similarity matrix in Line 9. Based on these thresholds, we declare for every row r in the similarity matrix all segments $\diamond \boldsymbol{o}_{r,l}^{\kappa_g}$ with $\boldsymbol{S}[r,l] \leq \bar{\zeta}_r$ to come from the same dynamic group.

Example 14: For Example 13, the similarity matrix for the trace $\boldsymbol{o_1}$ is

$$\boldsymbol{S} = \begin{bmatrix} 0.00 & 0.24 & 0.00 & 0.49 & 0.25 \\ 0.24 & 0.00 & 0.24 & 0.25 & 0.00 \\ 0.00 & 0.24 & 0.00 & 0.50 & 0.25 \\ 0.49 & 0.25 & 0.50 & 0.00 & 0.25 \\ 0.25 & 0.00 & 0.25 & 0.25 & 0.00 \end{bmatrix}. \tag{79}$$

Small values in the similarity matrix indicate similar segments. Thus, the diagonal is zero. Furthermore, the Segments 2 and 5 are considered similar. The same holds for Segments 3 and 6, because of their small DTW-metric. Choosing $\bar{\zeta}_i = 0.1$ for all rows, the following groups result: $\{1,3\}, \{2,5\}, \{4\}$.

FaMoS finds $\bar{\zeta}_i$ based on the minimum similarity value in the respective row i of the similarity matrix. A similarity threshold factor φ is used to scale the minimum similarity value. By choosing $\varphi > 1$, every segment is similar to at least one other segment. In addition, we limit $\bar{\zeta}_i$ to an interval $\left[\bar{\zeta}_{\min}, \bar{\zeta}_{\max}\right]$ with $\bar{\zeta}_{\max} \geq \overline{\zeta_{\min}}$ to avoid grouping too many or too few segments:

$$\bar{\zeta}_i = \begin{cases} \bar{\zeta}_{\max} & \text{if } \varphi \cdot \min\limits_{j \neq i} \boldsymbol{S}[i,j] > \bar{\zeta}_{\max} \\ \bar{\zeta}_{\min} & \text{if } \bar{\zeta}_{\min} > \varphi \cdot \min\limits_{j \neq i} \boldsymbol{S}[i,j] \\ \varphi \cdot \min\limits_{j \neq i} \boldsymbol{S}[i,j] & \text{otherwise.} \end{cases} \tag{80}$$

Based on the similarity matrix, groups are formed for each signal j in Line 9 of Listing 8.

Example 15: Figure 60 shows the grouping of segments for three traces $\boldsymbol{o_1}$, $\boldsymbol{o_2}$, and $\boldsymbol{o_3}$. The segments are grouped by the DTW-inspired comparison as shown

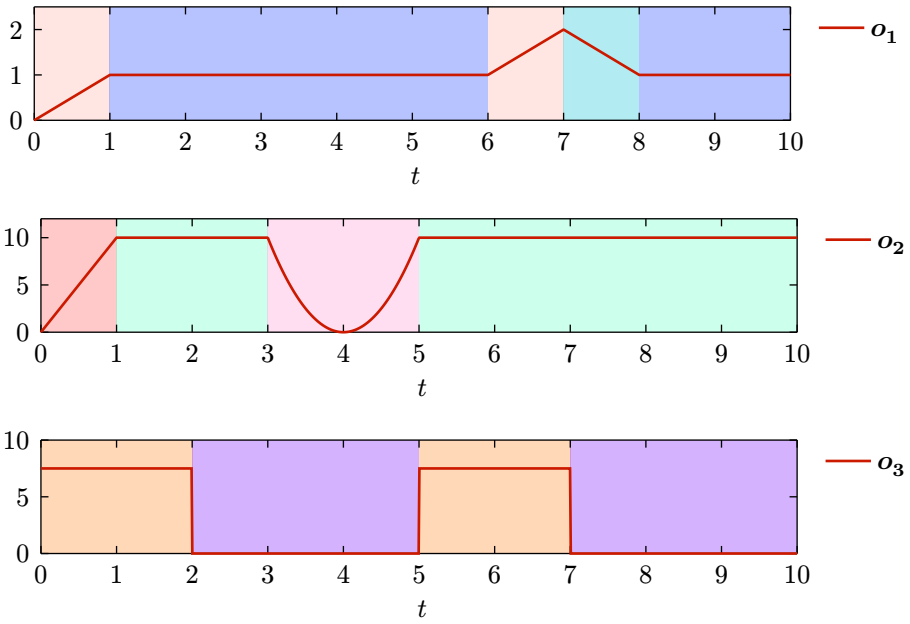


Figure 60 — Grouped segments for three traces o_1, o_2, and o_3. Every trace is grouped individually. The color indicates the group of the segment.

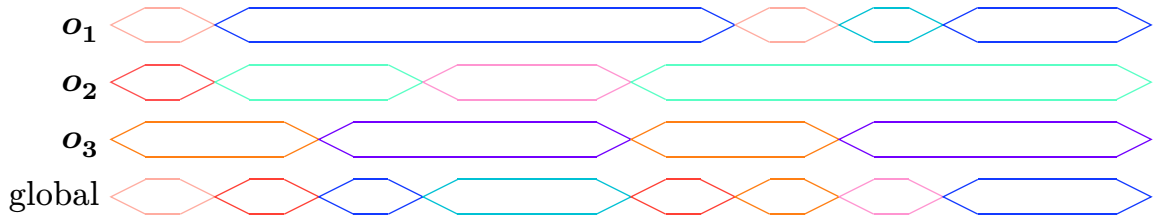


Figure 61 — Example of global grouping of segments. Global groups are formed by combining the group IDs of all variables. The example shows three signals with three groups each. The global groups are shown in the last row. The colors indicate the group IDs of the segments.

in Figure 59. Similar segments are grouped together, which is indicated by the same color.

The groups of individual signals are combined to global groups by combination of the groups for every timestep in Line 11 of Listing 8.

Example 16: Figure 61 shows an example of the global grouping of segments based on the results of Example 15. The local groups of the individual outputs are combined to global groups.

After the global grouping, an optional refinement step can be applied in Line 13 of Listing 8 to further improve the grouping and find similar groups. The DTW-based grouping depends on the choice of the parameter κ_g and the thresholds $\bar{\zeta}_i$. If some similarities are not detected by the DTW-inspired grouping, multiple groups may

contain segments with similar dynamics. To identify similar groups, FaMoS performs a comparison of the solution spaces of difference equations, i.e., a direct comparison of the dynamics represented by the ARX model of Equation 74. This is done with a Linear Matrix Inequality (LMI) formulation as used in [120]. A pair of preliminary groups (after DTW-inspired grouping) is compared using LMI. The two groups are combined to one group, if the solution space of their dynamics overlaps within a threshold ζ_{LMI}.

The DTW-inspired grouping is very fast while intuitively checking segment similarity. Solving the LMI formulation is computationally intensive and time-consuming. The proposed combination of the two approaches reduces the problem size for LMI. Thus, overall a moderate or small grouping effort is achieved.

6.3.3 Mode Characterization

After grouping, we reconstruct the system mode and characterize the dynamics of each mode. To do this, we identify the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ from Equation 74. We find these matrices by solving the problem

$$\min \parallel \boldsymbol{Y}'_{\mathcal{O}_q} - \boldsymbol{J}\boldsymbol{Y}_{\mathcal{O}_q} \parallel, \tag{81}$$

for each group g_q, where $\boldsymbol{Y}_{\mathcal{O}_q}$ and $\boldsymbol{Y}'_{\mathcal{O}_q}$ are the observation matrices on the set $\mathcal{O}_q = \{\diamondsuit \boldsymbol{s_1}, ..., \diamondsuit \boldsymbol{s_m}\}$ of trace segments $\diamondsuit \boldsymbol{s_i}$ of the considered group:

$$\boldsymbol{Y}_{\mathcal{O}_q} = \begin{bmatrix} \diamondsuit \boldsymbol{o_1^1}[1,...,l_1-1] & \diamondsuit \boldsymbol{o_1^2}[1,...,l_2-1] & \cdots & \diamondsuit \boldsymbol{o_1^m}[1,...,l_m-1] \\ \diamondsuit \boldsymbol{o_2^1}[1,...,l_1-1] & \diamondsuit \boldsymbol{o_2^2}[1,...,l_2-1] & \cdots & \diamondsuit \boldsymbol{o_2^m}[1,...,l_m-1] \\ \vdots & \vdots & & \vdots \\ \diamondsuit \boldsymbol{o_n^1}[1,...,l_1-1] & \diamondsuit \boldsymbol{o_n^2}[1,...,l_2-1] & \cdots & \diamondsuit \boldsymbol{o_n^m}[1,...,l_m-1] \\ \diamondsuit \boldsymbol{i_1^1}[1,...,l_1-1] & \diamondsuit \boldsymbol{i_1^2}[1,...,l_2-1] & \cdots & \diamondsuit \boldsymbol{i_1^m}[1,...,l_m-1] \\ \diamondsuit \boldsymbol{i_2^1}[1,...,l_1-1] & \diamondsuit \boldsymbol{i_2^2}[1,...,l_2-1] & \cdots & \diamondsuit \boldsymbol{i_2^m}[1,...,l_m-1] \\ \vdots & \vdots & & \vdots \\ \diamondsuit \boldsymbol{i_k^1}[1,...,l_1-1] & \diamondsuit \boldsymbol{i_k^2}[1,...,l_2-1] & \cdots & \diamondsuit \boldsymbol{i_k^m}[1,...,l_m-1] \end{bmatrix} \tag{82}$$

and

$$\boldsymbol{Y}'_{\mathcal{O}_q} = \begin{bmatrix} \diamondsuit \boldsymbol{o_1^1}[2,...,l_1] & \diamondsuit \boldsymbol{o_1^2}[2,...,l_2] & \cdots & \diamondsuit \boldsymbol{o_1^m}[2,...,l_m] \\ \diamondsuit \boldsymbol{o_2^1}[2,...,l_1] & \diamondsuit \boldsymbol{o_2^2}[2,...,l_2] & \cdots & \diamondsuit \boldsymbol{o_2^m}[2,...,l_m] \\ \vdots & \vdots & & \vdots \\ \diamondsuit \boldsymbol{o_n^1}[2,...,l_1] & \diamondsuit \boldsymbol{o_n^2}[2,...,l_2] & \cdots & \diamondsuit \boldsymbol{o_n^m}[2,...,l_m] \\ \diamondsuit \boldsymbol{i_1^1}[2,...,l_1] & \diamondsuit \boldsymbol{i_1^2}[2,...,l_2] & \cdots & \diamondsuit \boldsymbol{i_1^m}[2,...,l_m] \\ \diamondsuit \boldsymbol{i_2^1}[2,...,l_1] & \diamondsuit \boldsymbol{i_2^2}[2,...,l_2] & \cdots & \diamondsuit \boldsymbol{i_2^m}[2,...,l_m] \\ \vdots & \vdots & & \vdots \\ \diamondsuit \boldsymbol{i_n^1}[2,...,l_1] & \diamondsuit \boldsymbol{i_n^2}[2,...,l_2] & \cdots & \diamondsuit \boldsymbol{i_n^m}[2,...,l_m] \end{bmatrix}, \tag{83}$$

where l_i is the length of the segment $\diamondsuit \boldsymbol{s_i}$ in the group. The matrix $\boldsymbol{J} = [\boldsymbol{A}, \boldsymbol{B}]$ joins the state and input matrix of the state equation. The optimization problem in Equation 81 is solved for $\boldsymbol{J}$ with a least-squares approach. The solution of Equation 81

gives the optimal matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of the state equation from Equation 74 with respect to a quadratic minimization problem.

The result of the mode characterization is a set of flow functions $F = \left\{\langle q_1, f_1\rangle, ..., \langle q_{|G|}, f_{|G|}\rangle\right\}$, which holds a tuple of group ID and ARX model for each group $g_i, i \in \{1, ..., |G|\}$. Every group forms a dynamic mode in the final model.

6.3.4 Model Construction

The final step of FaMoS is model construction with decision trees. The model is constructed as a hybrid decision tree $\mathcal{T} = (X, V_H, E_H, Q, F)$, which combines the structure of a decision tree with the flow functions of the identified modes as in Definition 6.1. Using DTL, the decision tree represents the transitions between the modes of the system. The decision tree is constructed from the grouped segments from the previous step.

The system variables X are the input variables $\boldsymbol{i_1}, ..., \boldsymbol{i_k}$ and output variables $\boldsymbol{o_1}, ..., \boldsymbol{o_d}$ of the system. Every group g of segments corresponds to a mode of the system, i.e., we find $Q = \{q_1, ..., q_{|G|}\}$. Further, we extend every trace $\boldsymbol{s}$ by a trace of modes $\boldsymbol{q}$, which gives the mode ID of the system at every sample.

The decision tree with nodes V_H and edges E_H is found via DTL. The learned decision tree represents transitions between modes based on the current values of the system variables. Every leaf in the decision tree points to a system mode and is associated with its respective flow function from the set F of flow functions found in the mode characterization step. Thus, the feature vector for DTL consists of the current mode $\boldsymbol{q}[t]$, the current output variables $\boldsymbol{o_1}[t], ..., \boldsymbol{o_n}[t]$, and the current input variables $\boldsymbol{i_1}[t], ..., \boldsymbol{i_k}[t]$. The class label is the next mode $\boldsymbol{q}[t+1]$. The set of observations $\mathcal{O}_{\text{DTL}}$ for DTL is constructed from the set $\mathcal{O}_{\text{trace}}$ of traces as follows

$$\begin{aligned}\mathcal{O}_{\text{DTL}} = \bigcup_{\boldsymbol{s} \in \mathcal{O}_{\text{trace}}} \{\langle [\boldsymbol{q}[t], \boldsymbol{o_1}[t], ..., \boldsymbol{o_n}[t], \boldsymbol{i_1}[t], ..., \boldsymbol{i_k}[t]], \boldsymbol{q}[t+1]\rangle, \\ t \in [1, ..., |\boldsymbol{s}| - 1]\},\end{aligned} \tag{84}$$

where $\boldsymbol{q}[t]$ gives the mode of the system at time t and $\boldsymbol{q}[t+1]$ gives the mode in the next timestep $t+1$.

6.3.5 Evaluation

Setup

Examples:

- Boiler: the controlled temperature range scenario of the boiler example from Section 3.7.4
- Power Converter: buck converter variant of the power converter example from Section 3.7.5

- Two Tank: two tank system example from Section 3.7.3.

Language: Matlab

Learning Set:

- Boiler and Power Converter: 8 traces per example
- Two Tank: 1 trace

Test Set:

- Boiler and Power Converter: 2 traces per example
- Two Tank: 1 trace

Metrics:

- Time: The time needed for the individual steps of the learning process. The time is measured in seconds.
- MSE: Mean-square error of the predicted trace to the ground truth trace.
- Ω_{seg}: Segmentation objective function from Equation 72, which determines the mean absolute error of the predicted transition points to the ground truth transition points. Additionally, a penalty term is added for the divergence in the number of predicted and ground truth transition points.
- Ω_{group}: Grouping objective function from Equation 73, which is the mean error over the groups, where the error e_g of a group is the sum over the MSE of all variables and all segments of the group. An additional penalty factor is added for the divergence in the number of predicted and ground truth groups.

Device: IntelCore i7-9750H CPU@2.60GHz, 16GB RAM

We evaluate FaMoS on a set of example systems. The evaluation covers all steps of the learning process, i.e., trace segmentation, segment grouping, mode characterization, and model construction. In addition, we compare the results of FaMoS to HAutLearn [120], a state-of-the-art approach for learning hybrid automata models from observed traces.

HAutLearn identifies transition points in the same way as FaMoS, i.e., by using a sliding window approach. The grouping in HAutLearn is done with LMIs assuming that the dynamics are represented by matrix differential equations. Consequently, mode characterization uses the same method as FaMoS, i.e., identifies the matrices of the differential equations. Finally, a hybrid automaton is constructed via a PTA using the RANSAC algorithm [160]. This algorithm has three parameters: 1) the error tolerance, ψ, to check the compatibility of a transition point with a model candidate, 2) the number of iterations η, for each model estimation, and 3) the threshold number of compatible points, γ, which indicate a valid estimation.

All parameters of FaMoS are listed in Table XII and the parameters used for the examples in this section are shown in Table XIII. Parameterization of FaMoS is done manually and based on S. Plambeck et al. [34]. The parameters of HAutLearn used for

Table XII — Parameters for the FaMoS algorithm.

Step	Parameter	Description
Trace Segmentation	ω	Window width
Trace Segmentation	ω_{size}	Minimum distance of transition points
Trace Segmentation	κ_{seg}	Maximum order of derivative
Grouping	κ_g	Order of derivative
Grouping	$\varphi, \bar{\zeta}_{\text{min}}, \bar{\zeta}_{\text{max}}$	Similarity thresholds
Grouping	ζ_{LMI}	LMI tolerance
State Equation	κ	Order of difference equations

Table XIII — Parameterization of FaMoS for all examples.

Example	General	Segmentation			Grouping				
	t_S [s]	κ_{seg}	ω	ω_{size}	ζ_{LMI}	κ	φ	$\bar{\zeta}_{min/max}$	κ_g
Boiler	$1.0 \cdot 10^{-2}$	3	10	10	$1.5 \cdot 10^{-3}$	1	2.5	$0.01, 0.1$	0
Power Converter	$1.0 \cdot 10^{-5}$	1	10	10	$2.0 \cdot 10^{-6}$	1	2.5	$0.03, 1.0$	0
Two Tank	0.1	1	30	30	$1.0 \cdot 10^{-4}$	1	5	$1 \cdot 10^{-4}, 10$	0

Table XIV — Parameterization of HAutLearn for all examples.

Example	η	γ	ψ
Boiler	10^5	10	$2.7 \cdot 10^{-3}$
Power Converter	10^5	10	$2.0 \cdot 10^{-4}$
Two Tank	10^5	1	$2.0 \cdot 10^{-4}$

the examples are shown in Table XIV. The LMI parameters of FaMoS and HAutLearn are the same.

The learned flow functions of FaMoS and HAutLearn are shown in Table XV. A timeout of 5 hours is set for each example and '-' indicates that a timeout happened. FaMoS finishes within 5 hours for all examples while HAutLearn finishes within 5 hours for the boiler example and the two tank example but not for the power converter. For the boiler example, FaMoS finds two modes, which represent the heating and cooling mode of the boiler. HAutLearn finds the same two modes, but additionally a third mode, which is not present in the ground truth as the algorithm fails to merge modes in the segment grouping step.

For the power converter example, FaMoS finds three modes, which is equivalent to the number of modes in the ground truth model of Figure 25. The ground truth model of the two tank example from Equation 10 has three modes. Mainly the first and the last mode are active, as the height of the water level in the first tank is usually

Table XV — Learned flow functions of FaMoS and HAutLearn. A '-' indicates a timeout of 5 hours for the learning process.

Example	Approach		Learned Flow Functions
Boiler	FaMoS	Group 1 —	$x[t] = 1.008 \cdot x[t-1] - 0.003$
		Group 2 —	$x[t] = 1.001 \cdot x[t-1] - 0.005$
	HAutLearn	Group 1 —	$x[t] = 1.008 \cdot x[t-1] - 0.003$
		Group 2 —	$x[t] = 1.001 \cdot x[t-1] - 0.005$
		Group 3 —	$x[t] = 0.984 \cdot x[t-1] + 0.018$
Power Converter	FaMoS	Group 1 —	$\begin{bmatrix} x_1[t] \\ x_2[t] \end{bmatrix} = \begin{bmatrix} 0.998 & -0.004 \\ 0.005 & 1.000 \end{bmatrix} \cdot \begin{bmatrix} x_1[t-1] \\ x_2[t-1] \end{bmatrix} + \begin{bmatrix} 0.003 \\ 0.0 \end{bmatrix}$
		Group 2 —	$\begin{bmatrix} x_1[t] \\ x_2[t] \end{bmatrix} = \begin{bmatrix} 0.998 & -0.004 \\ 0.005 & 1.000 \end{bmatrix} \cdot \begin{bmatrix} x_1[t-1] \\ x_2[t-1] \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$
		Group 3 —	$\begin{bmatrix} x_1[t] \\ x_2[t] \end{bmatrix} = \begin{bmatrix} 0.956 & 0.0 \\ 0.0 & 1.000 \end{bmatrix} \cdot \begin{bmatrix} x_1[t-1] \\ x_2[t-1] \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$
	HAutLearn		-
Two Tank	FaMoS	Group 1 —	$\begin{bmatrix} h_1[t] \\ h_2[t] \end{bmatrix} = \begin{bmatrix} 0.992 & 0.002 \\ 0.009 & 0.991 \end{bmatrix} \cdot \begin{bmatrix} h_1[t-1] \\ h_2[t-1] \end{bmatrix} + \begin{bmatrix} 0.006 \\ 0.002 \end{bmatrix}$
		Group 2 —	$\begin{bmatrix} h_1[t] \\ h_2[t] \end{bmatrix} = \begin{bmatrix} 1.0 & 0.0 \\ -0.027 & 0.996 \end{bmatrix} \cdot \begin{bmatrix} h_1[t-1] \\ h_2[t-1] \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.027 \end{bmatrix}$
	HAutLearn	Group 1 —	$\begin{bmatrix} h_1[t] \\ h_2[t] \end{bmatrix} = \begin{bmatrix} 1.110 & 0.031 \\ -0.506 & 0.864 \end{bmatrix} \cdot \begin{bmatrix} h_1[t-1] \\ h_2[t-1] \end{bmatrix} + \begin{bmatrix} -0.133 \\ 0.604 \end{bmatrix}$
		Group 2 —	$\begin{bmatrix} h_1[t] \\ h_2[t] \end{bmatrix} = \begin{bmatrix} 0.994 & 0.0 \\ -0.001 & 0.998 \end{bmatrix} \cdot \begin{bmatrix} h_1[t-1] \\ h_2[t-1] \end{bmatrix} + \begin{bmatrix} 0.006 \\ 0.003 \end{bmatrix}$
		Group 3 —	$\begin{bmatrix} x_1[t] \\ x_2[t] \end{bmatrix} = \begin{bmatrix} 1.518 & 0.117 \\ -2.343 & 0.474 \end{bmatrix} \cdot \begin{bmatrix} x_1[t-1] \\ x_2[t-1] \end{bmatrix} + \begin{bmatrix} -0.614 \\ 2.775 \end{bmatrix}$

higher than the height of the water level in the second tank, i.e., $h_1 > h_2$. FaMoS finds two modes, which represent the first and the last mode of the ground truth model. Nevertheless, a direct comparison to the ground truth is not possible as the ground truth differential equation is not linear and the learned linear differential equations are only approximations of the ground truth.

Table XVI presents the accuracy and performance results for system identification using FaMoS (FaM) and HAutLearn (HAut). Both approaches demonstrate very similar performance on the boiler example and the two tank example. Since both methods employ the same segmentation approach, only the segmentation objective function Ω_{seg} from the FaMoS run is reported. For the grouping step, FaMoS achieves marginally better results than HAutLearn on the boiler example, which is somewhat unexpected given that HAutLearn employs a dynamics-based approach with LMIs instead of a

Table XVI — Evaluation metrics of FaMoS (FaM) and HAutLearn (HAut) consisting of the segmentation objective function Ω_{seg}, the grouping objective function Ω_{group}, and the MSE of the normalized predicted traces to the ground truth traces.

Example	Ω_{seg}	Ω_{group}		MSE	
		FaM	HAut	FaM	HAut
Boiler	3.06	0.003	0.007	0.002	0.003
Power Converter	4.96	$8.6 \cdot 10^{-5}$	-	$[30.6 \;\; 1.4]^T \cdot 10^{-5}$	-
Two Tank	3.06	$1.00 \cdot 10^{-4}$	$1.14 \cdot 10^{-5}$	$[0.006 \;\; 0.06]^T$	$[0.007 \;\; 0.181]^T$

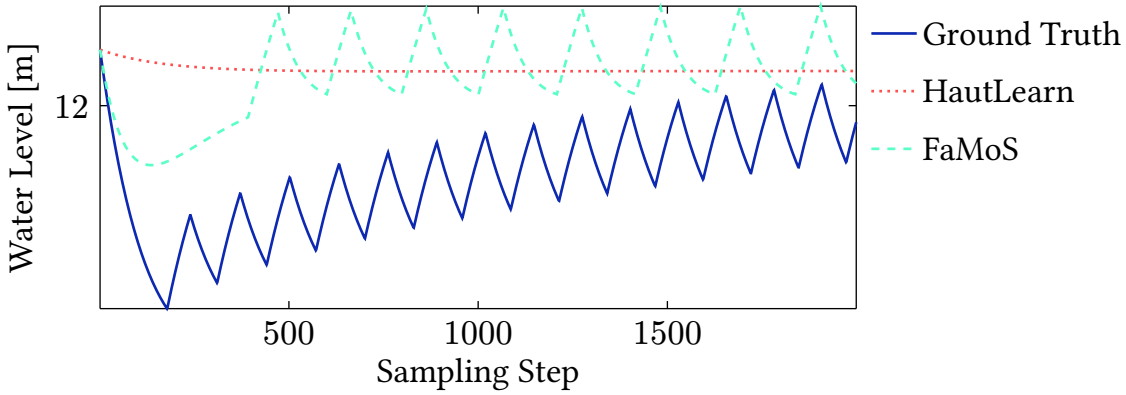
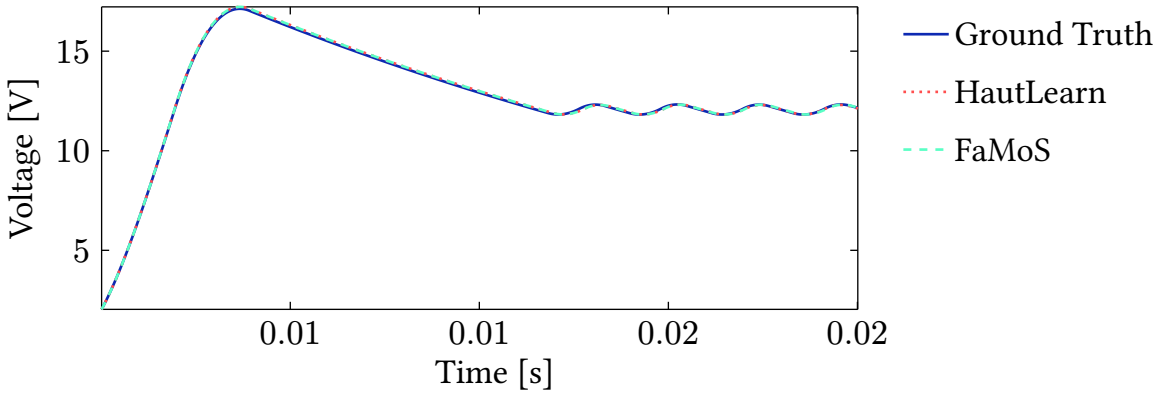


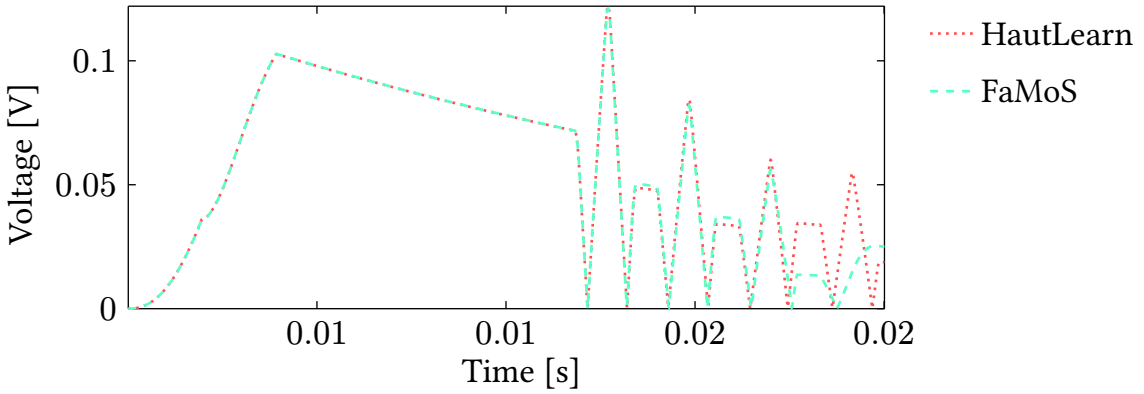
Figure 62 — Predicted traces of the two tank system for HAutLearn (dotted-red) and FaMoS (dashed-green) in comparison to the ground truth trace (blue).

purely similarity-based approach with DTW in FaMoS. However, the difference is minimal, and both approaches achieve good values for the grouping objective function Ω_{group}. Regarding the MSE, FaMoS achieves comparable or slightly better results than HAutLearn. FaMoS predicts system mode changes by applying the decision function of the learned decision tree model to the feature vector of each sample. HAutLearn predicts the next mode based on active event conditions and the current system mode using the learned hybrid automaton.

We observe that the MSE is similar to the grouping objective function Ω_{group} for the boiler example and the power converter example. This shows the generalization capability of the learned models as the grouping objective function Ω_{group} is determined on the learning set, while the MSE is determined on the test set. For the two tank example, the MSE is significantly larger than the grouping objective function Ω_{group}. This difference is due to a bad generalization of the learned model to the test set, which has different initial conditions than the learning set. The predicted traces for the two tank example are shown in Figure 62. We observe that FaMoS still represents the system dynamics well, but fails to predict the system mode changes correctly. HAutLearn fails



(a) Predicted traces for the converter for HAutLearn (dotted-red) and FaMoS (dashed-green) in comparison to the ground truth trace (blue).



(b) Error traces for the converter for HAutLearn (dotted-red) and FaMoS (dashed-green).

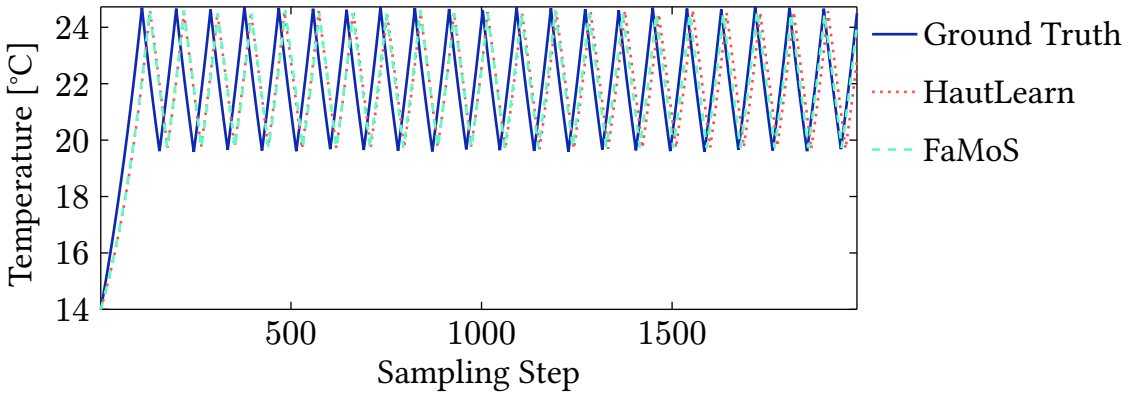
Figure 63 — Predicted traces for the converter for HAutLearn (dotted-red) and FaMoS (dashed-green) in comparison to the ground truth trace (blue).

to predict the system mode changes completely, resulting in a bad accuracy of the predicted trace.

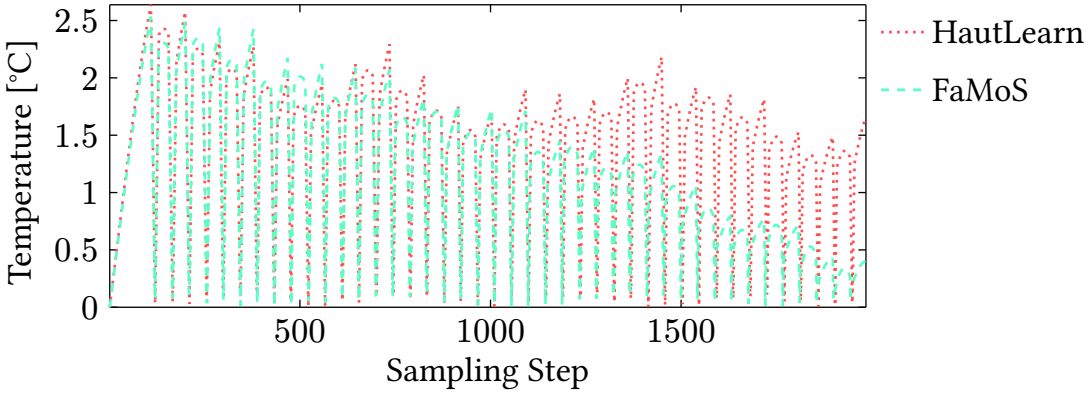
Figure 63 shows the voltage (variable x_2) of the power converter example for the predicted traces against the ground truth trace as well as the error of the predicted trace over time on one of the evaluation traces. HAutLearn and FaMoS both generate predictions that are close to the ground truth. Especially for larger t, the predictions of FaMoS are closer to the ground truth than the predictions of HAutLearn.

Figure 64 shows the temperature of the boiler example for the predicted traces against the ground truth trace as well as the error of the predicted trace over the sample time. Here, the prediction error is comparably large for both approaches, which mainly results from a time shift of the predicted trace. Again, FaMoS achieves a smaller prediction error for larger t.

The execution times of the individual steps of FaMoS and HAutLearn are shown in Table XVII. The segmentation time is the same for both approaches as they use the same segmentation method. The grouping time of FaMoS is significantly lower than the grouping time of HAutLearn. Additionally, the time for the grouping step of



(a) Predicted traces for the boiler for HAutLearn (dotted-red) and FaMoS (dashed-green) in comparison to the ground truth trace (blue).



(b) Error traces for the boiler for HAutLearn (dotted-red) and FaMoS (dashed-green).

Figure 64 — Predicted traces for the boiler for HAutLearn (dotted-red) and FaMoS (dashed-green) in comparison to the ground truth trace (blue).

Table XVII — Execution times of the individual steps of FaMoS and HAutLearn in seconds. The segmentation and mode characterization is the same for all methods. Thus, only the time of the FaMoS run is given. The extraction method is the same for both variants of FaMoS, thus, only the time for the FaMoS run is shown. A timeout of 5 hours is set for each example; '-' indicates that a timeout happened.

Example	Segmentation Time [s]	Grouping Time [s]			Characterization Time [s]	Extraction Time [s]	
		FaM	FaM + LMI	HAut		FaM	HAut
Boiler	0.54	16.04	16.49	408.04	0.17	0.40	36.95
Power Converter	0.40	5.48	12.50	-	0.01	0.04	-
Two Tank	0.13	0.86	0.94	52.77	0.02	0.24	1.51

FaMoS with the optional refinement using LMIs is shown. Adding the refinement step increases the execution time of the grouping step only slightly as LMI is applied on a small number of pre-grouped segments. For characterization, FaMoS and HAutLearn

use the same method, which is based on the solution of a least-squares problem of the state equation. The extraction time of FaMoS is significantly lower than the extraction time of HAutLearn. HAutLearn uses a PTA to extract the hybrid automaton model, which is computationally intensive and time-consuming. The extraction time of FaMoS uses a lightweight DTL approach.

Overall, the DTW-based approach of FaMoS provides accurate results with very low execution time. FaMoS outperforms the HAutLearn framework in runtime and at least matches the HAutLearn framework in prediction accuracy for the presented example systems.

6.4 Symbolic Regression for Hybrid Decision Tree Learning

FaMoS is a fast and efficient way to learn hybrid decision tree models from observed traces. Nevertheless, the approach is limited to systems whose dynamics are represented by ARX models. The approach is not able to learn models with other, e.g., more complex flow functions, such as nonlinear flow functions or higher order differential equations. Furthermore, the segmentation and grouping of signals uses similarities in the signals and does not consider the actual dynamics. This is counterintuitive as for an unknown, black-box system, the dynamics of the system should be the main criterion for the segmentation and grouping of segments. A new dynamic mode should be created only if the dynamics of the system change in a way such that they cannot be represented by the same flow function. Otherwise, the segmentation and grouping of segments may lead to a large number of modes, which are not necessary for representing the system.

To overcome these limitations, we propose a new approach to learn hybrid decision tree models from observed traces of a system. The approach is based on Symbolic Regression (SR) as introduced in Section 3.6 and is published by S. Plambeck et al. [36] [35].

The identification of a hybrid decision tree with symbolic regression systematically implements the learning process, as depicted in Figure 54. The initial steps — trace segmentation and segment grouping — utilize symbolic regression techniques. Mode characterization is inherently captured within the symbolic regression results obtained during grouping. The final step, model construction, leverages a hybrid decision tree as in Definition 6.1, following an approach similar to FaMoS, to synthesize the complete model.

The symbolic regression approach for hybrid decision tree learning identifies one-dimensional flow functions for each mode of the hybrid decision tree in the form of a symbolic expression:

$$o = f(i_1, ..., i_k, s_1, ..., s_m), \tag{85}$$

where o is the output variable, $i_1, ..., i_k$ are the input variables, and $s_1, ..., s_m$ are the state variables of the system. The learning process starts with a trace $\boldsymbol{s}$ observed on the system, which consists of the subtraces for the individual signals, i.e.,

$$\boldsymbol{s} = \begin{bmatrix} \boldsymbol{o} \\ \boldsymbol{i_1} \\ \vdots \\ \boldsymbol{i_k} \\ \boldsymbol{s_1} \\ \vdots \\ \boldsymbol{s_m} \end{bmatrix}. \tag{86}$$

In the following, we describe the individual steps of the symbolic regression approach for hybrid decision tree learning starting with trace segmentation, followed by segment grouping including mode characterization, and model construction.

6.4.1 Trace Segmentation

The trace segmentation step identifies transition points in the observed trace of the system. At the transition points, the system changes its mode, i.e., the dynamics of the system change. Listing 9 describes the procedure used to detect transition points in the trace of the system.

The algorithm starts with a fixed initial window $\boldsymbol{s_w}$ of size l_{init} in Line 8. The window is extended to the right by a step size l_{step} in Line 11 until the samples in the window cannot be described by the same symbolic expression anymore. This is detected by evaluating a segmentationCriterion(e, e_{prev}) on the error e and e_{prev} of the symbolic expression on the samples of the current and previous window in Line 7 of Listing 9. The segmentation criterion is a user-defined function that determines whether the current window can still be described by the same symbolic expression. If the segmentation criterion is fulfilled, the window is extended to the right by l_{step}. As soon as the segmentation criterion is not fulfilled anymore, a transition point is detected.

Here, the segmentation criterion of Listing 10 is applied. If the error e is smaller than the stagnation threshold ξ_S, the current window is still considered to be part of the same mode. The current window is also considered to be part of the same mode if the error e is larger than the stagnation threshold, but smaller than the previous error e_{prev}. If the segmentation criterion is not fulfilled anymore, a transition point is detected. The identified window is stored as a segment in the set S in Line 14. The symbolic expression is learned incrementally in Line 10, which performs η_{update}-many iterations of SR to adapt the current expression to the extended window.

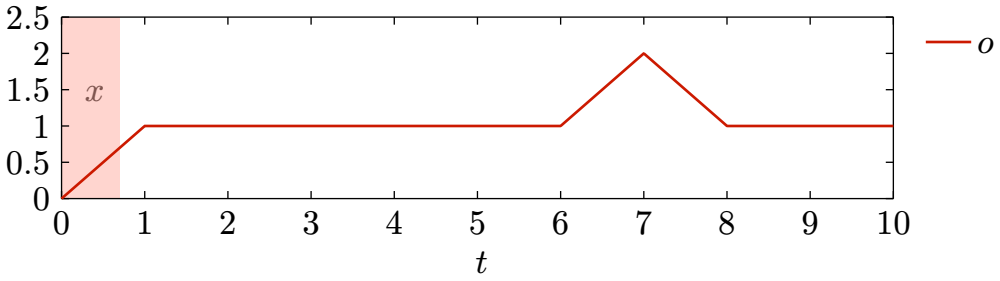
Whenever a transition point is found, learning restarts and creates a new window starting at the end of the previously identified window. For each segment, a new expression is learned from scratch.

```
1:  function detectTransitionPoints(s)
2:      Hyperparameters: l_init, l_step, η_init, η_update,
            segmentationCriterion(·,·), ψ_S, η_S, B_S
3:
4:      i_start ← 0, i_end ← l_init, η_init ← η_init
5:      while i_end < |s| do
6:          e_prev, e ← 0
7:          while segmentationCriterion(e, e_prev) ∨ e_prev ≡ 0 do
8:              s_w ← s[i_start, min(i_end, |s|)]
9:              e_prev ← e
10:             f, e ← learnExpression(s_w, η)
11:             i_end ← i_end + l_step
12:             η ← η_update
13:         end
14:         S ← S ∪ {s_w[0, end − l_step]}
15:         i_start ← i_end − l_step, i_end ← i_start + l_init, η ← η_init
16:     end
17:     return S
18: end
```

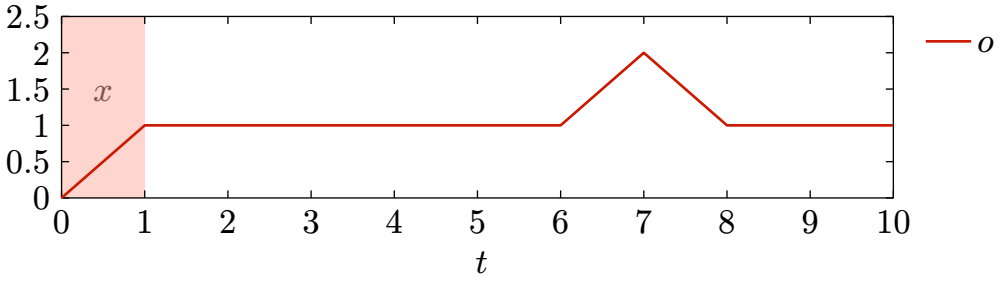
Listing 9 — Detection of transition points with SR. The algorithm considers a window of the trace and extends this window until the segmentation criterion is not fulfilled anymore. A transition point is detected, and the window is stored as a segment.

```
1:  function segmentationCriterion(e, e_prev)
2:      Hyperparameter: ξ_S
3:      if (e < ξ_S) ∨ (e ≤ e_prev) then
4:          return true
5:      end
6:      else
7:          return false
8:      end
9:  end
```

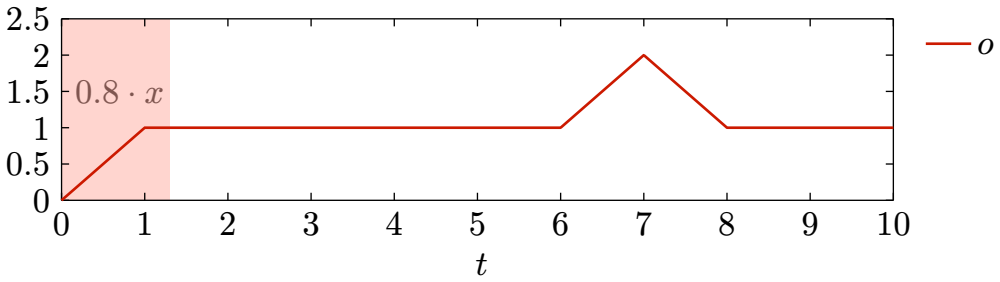
Listing 10 — The segmentation criterion of the segmentation step compares the current error e with the stagnation threshold ξ_S and the previous error e_{prev}. If the current error is smaller than the stagnation threshold or smaller than the previous error, the segmentation criterion is fulfilled.



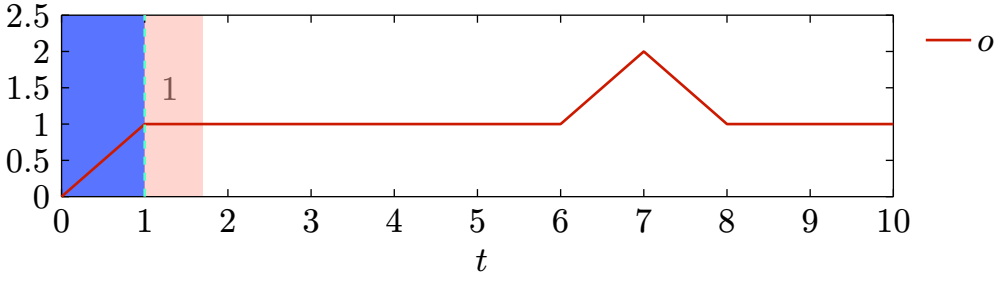
(a) First iteration of the trace segmentation algorithm. The initial window is shown. The learned expression for this window is x. The segmentation criterion is fulfilled, thus, the window is extended.



(b) Second iteration of the trace segmentation algorithm. The window is extended to the right by l_{step} and the learned expression for this window is x. The segmentation criterion is fulfilled, thus, the window is extended again.



(c) Third iteration of the trace segmentation algorithm. The window is extended to the right by l_{step} and the learned expression for this window is $0.8 \cdot x$. The segmentation criterion is not fulfilled anymore, thus, a transition point is detected.



(d) The next window is initialized at the end of the previous segment. The learned expression for this window is 1. The segmentation criterion is fulfilled, thus, the window is extended.

Figure 65 — Excerpt of trace segmentation with SR on an example trace.

Example 17: Figure 65 illustrates the trace segmentation process using symbolic regression. Figure 65a shows an initial window of the trace with a candidate expression x learned from the samples in the window. In Figure 65b the window is extended as the segmentation criterion is fulfilled and the candidate expression is still x. The segmentation criterion is fulfilled, thus, the window is extended to the right by l_{step} in Figure 65c. The candidate expression is updated to $0.8 \cdot x$ and the segmentation criterion is not fulfilled as the error of the candidate expression is too large. In Figure 65d, a transition point is detected, and a new window is created starting at the end of the previous segment. The candidate expression for the new window is 1.

The following list summarizes the hyperparameters of the trace segmentation step:

- l_{init}: the initial window size when learning an expression from scratch,
- l_{step}: the step-width for extending the window in the inner loop,
- η_{init}: the number of iterations of SR when learning an expression from scratch,
- η_{update}: the number of iterations of SR for updating the learned expression on an extended window,
- segmentationCriterion(e, e_{prev}): the criterion to determine whether a transition point is reached.
- ξ_S: A stagnation-threshold for the segmentation criterion.
- ψ_S: the parsimony coefficient of SR,
- η_S: the number of individuals in the population of SR,
- B_S: the set of basic operators and functions for SR.

The advantage of the symbolic regression approach for trace segmentation is that it does not require any prior knowledge about the type of dynamics of the system except for the basic operators and functions used in SR. The symbolic regression approach is able to learn the dynamics of the system from the observed traces. By this, the segmentation is based on the actual dynamics of the system. Transition points are identified when the candidate expression fails to achieve sufficient accuracy within a given window, indicating a change in the system dynamics.

6.4.2 Segment Grouping & Mode Characterization

The next step of the symbolic regression approach for hybrid decision tree learning is segment grouping, which groups segments that are described by the same flow function. The grouping is done with the help of SR, which learns a symbolic expression for groups of segments. By this, the grouping directly characterizes the modes of the hybrid decision tree. Listing 11 presents the pseudocode for segment grouping using SR. The input to the grouping is the set of detected segments S from the previous step. Output of the algorithm are two maps G of groups and F of learned expressions for each group, respectively.

```
1:  function groupSegments(S)
2:      Hyperparameters: η_group, groupingCriterion(·,·), ψ_G, η_group, B_G
3:
4:      g ← S[0]
5:      S ← S \ g
6:      G ← {⟨0, g⟩}
7:      γ ← 1
8:      for ◇s ∈ S do
9:          b_found ← false
10:         for ⟨γ, g⟩ ∈ G do
11:             f, e ← learnExpression(◇s ∪ g, η_group)
12:             if groupingCriterion(◇s, g) then
13:                 g ← s ∪ g
14:                 F ← ⟨γ, f⟩
15:                 γ ← γ + 1
16:                 b_found ← true
17:                 break
18:             end
19:         end
20:         if ¬b_found then
21:             G ← G ∪ {⟨|G|, ◇s⟩}
22:         end
23:     end
24:     return G, F
25: end
```

Listing 11 — Grouping of segments with SR. The algorithm iterates over all segments and groups them with the help of symbolic regression. The segments are grouped if the groupingCriterion() is fulfilled. If no group is found, a new group is created.

The first segment $\diamond \boldsymbol{s}$ in the set S is used to initialize the first group in Line 4 of Listing 11. Afterward, we iterate for every segment in S over all known groups with the help of the for-loop in Line 5 and Line 7. An expression is learned on the conjunction of the current segment $\diamond \boldsymbol{s}$ and the current group g in Line 8 using SR. If the learned expression fulfills the groupingCriterion() in Line 9, the current segment is added to the current group g.

The groupingCriterion() is a user-defined function that determines whether the segment and the group can be described by the same symbolic expression. Here, we use the grouping criterion from Listing 12. This criterion evaluates whether the error of the learned expression on the conjunction of the segment and the group is smaller than the error of the learned expression on the group alone multiplied by a relaxation

```
1:  function groupingCriterion(◇s, g)
2:      Hyperparameter: ξ_G
3:      f_join, e_join ← learnExpression(◇s ∪ g, η_group)
4:      f_group, e_group ← learnExpression(g, n_group)
5:      if e_join ≤ ξ_G · e_group then
6:          return true
7:      end
8:      else
9:          return false
10:     end
11: end
```

Listing 12 — The grouping criterion of the grouping step learns expressions for the conjunction of the current segment $\diamond \boldsymbol{s}$ and the current group g and evaluates whether the error of the learned expression on the conjunction is smaller than the error of the learned expression on the group alone multiplied by a relaxation parameter ξ_G.

parameter ξ_G. With a relaxation parameter $\xi_G \geq 1$, the groupingCriterion() allows for a certain increase in the error of the learned expression on the conjunction of the segment and the group.

If no matching group is found for the current segment, the segment forms a new group, which is added to the set of groups G in Line 17 of Listing 11.

The following list summarizes the hyperparameters of the segment grouping step:

- η_{group}: the number of iterations of SR for learning an expression on combined segments,
- groupingCriterion(): the criterion to determine whether a segment is joined with a group.
- ξ_G: the relaxation parameter of the grouping criterion,
- ψ_G: the parsimony coefficient of SR,
- p_G: the number of individuals in the population of SR,
- B_G: the set of basic operators and functions for SR.

The symbolic regression approach for segment grouping is able to group segments that are described by the same flow function. The grouping is based on the actual dynamics of the system and does not require any prior knowledge about the type of dynamics of the system except for the basic operators and functions used in the symbolic regression.

6.4.3 Model Construction

The final step of the symbolic regression approach for hybrid decision tree learning is model construction. The model construction step uses a hybrid decision tree

$\mathcal{T} = (X, V_H, E_H, Q, F)$ as defined in Definition 6.1 to construct a model from the grouped segments and the learned flow functions.

The system variables X are the input variables $i_1, ..., i_k$, the output variable o, and the state variables $s_1, ..., s_m$. Every group corresponds to a mode of the hybrid decision tree and, thus, we define $Q = \left\{q_1, ..., q_{|G|}\right\}$, where q_j is the mode ID of group g_j. Further, a mode is associated with a flow function f. The tuples $\langle q_j, f_j \rangle$ build the set of flow functions F.

The decision tree with nodes V_H and edges E_H is constructed from samples of the groups. The feature vector of the decision tree is the vector of the variables in X and the class label is the current mode q. Every leaf of the decision tree refers to a mode of the hybrid decision tree and is associated with a flow function f. The learning set for DTL is constructed as follows

$$\mathcal{O}_{\text{DTL}} = \{\langle g[j], q \rangle : \langle g, q \rangle \in G, j \in \{0, ..., |g| - 1\}\}, \tag{87}$$

where $g[j]$ gives the values of all variables in X for sample j of group g.

6.4.4 Evaluation

Setup

Examples:

- Boiler: the controlled temperature range scenario of the boiler example from Section 3.7.4. Symbolic regression for the trace segmentation and grouping uses the operators addition, subtraction, and multiplication. The target variable $\dot{x}$ is numerically calculated from x.
- Power converter: generic variant of the power converter example from Section 3.7.5. In the learning steps for segmentation and grouping, we use the operators addition, subtraction, multiplication, and division as well as the square-root as basic operators for SR. The to be learned derivative $\dot{w}_2$ is numerically calculated from the data. Feature variables are w_1, w_2, and the continuous time t.
- Two tank: two tank system example from Section 3.7.3. The operators for SR in the segmentation and grouping step are addition, subtraction, multiplication, and division as well as the square-root. The variables used for learning are the inflow Q_p, the height of the two tanks h_1 and h_2 as well as the constants C_b and A. For DTL, the variable V_b is used as an additional feature.

Language: Python, using the framework PySR [161] for SR

Scenarios: For the two tank example, we use two scenarios:

- Two tank sub.: the term $h_{\text{sqr}} = \sqrt{|h_1 - h_2|}$ is pre-calculated on the data and used as an additional variable to learn the derivative $\dot{h}_1$. Practically,

Table XVIII — Parameters for system identification with SR for all examples.

Example	l_{init}	l_{step}	η_{init}	η_{update}	ξ_S	η_{group}	ξ_G	ψ_G	ψ_S	p_G
Boiler	30	1	20	5	$1 \cdot 10^{-4}$	20	1.6	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	31
Power Converter	100	20	100	20	$1 \cdot 10^{-6}$	270	0.65	0.003	0.865	20
Two Tank *sub*	40	1	150	25	$1.28 \cdot 10^{-8}$	200	0.79	0.450	0.760	15
Two Tank *w/o sub*	50	10	200	5	$1 \cdot 10^{-6}$	100	1.25	0.850	0.110	40

providing such pre-calculated terms is partial knowledge about the physical system and the operators and terms that are relevant for its behavior.

- Two tank w/o sub.: the term h_{sqr} is not pre-calculated and the derivative $\dot{h}_1$ is learned without this term.

Learning Set: 1 trace per example

Test Set: 1 trace per example

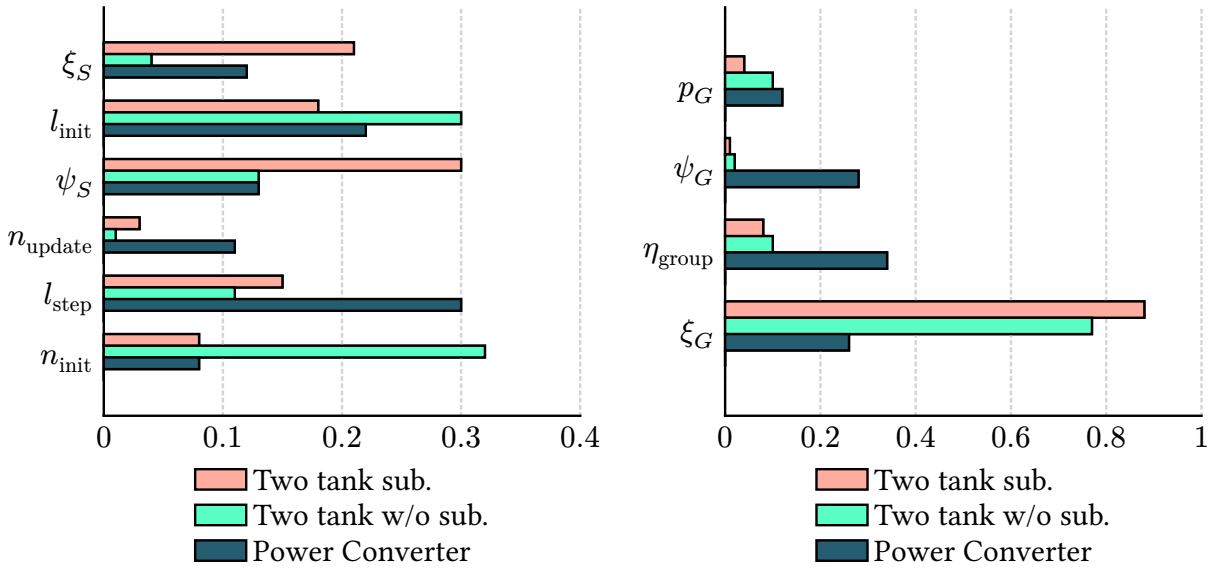
Metrics:

- MSE: Mean-square error of the predicted trace to the ground truth trace.
- Ω_{seg}: Segmentation objective function from Equation 72, which determines the mean absolute error of the predicted transition points to the ground truth transition points. Additionally, a penalty term is added for the divergence in the number of predicted and ground truth transition points.
- Ω_{group}: Grouping objective function from Equation 73, which is the mean error over the groups, where the error e_g per group is the loss of the SR of group g. An additional penalty factor is added for the divergence in the number of predicted and ground truth groups.
- Time: The time needed for the individual steps of the learning process. The time is measured in seconds.

Device: IntelCore i7-9750H CPU@2.60GHz, 16GB RAM

We evaluate the approach on a set of example systems and scenarios. The evaluation covers all steps of the learning process. The parameters for the segmentation and grouping algorithms for the examples and scenarios are shown in Table XVIII. A hyperparameter optimization is done with Optuna [162], which implements an efficient sampling of the parameter space using the objective functions of Equation 72 and Equation 73.

Parameter optimization with Optuna finds parameter importances using the fANOVA approach [163], which identifies the most influential hyperparameters by analyzing their impact on the objective functions via random forests, an ensemble method integrating multiple decision trees. Figure 66 shows the parameter importances



(a) Segmentation (b) Grouping

Figure 66 — Parameter importances of the two tank system with substitution (green), two tank system without substitution (red), and power converter (blue).

Table XIX — Learned expressions per group for the examples.

Example		Learned Flow Function
Boiler	Group 1 —	$4.83 \cdot 10^{-3} \cdot x$
	Group 2 —	$-5.03 \cdot 10^{-3} \cdot x$
Power Converter	Group 1 —	$0.003 - 0.005 w_1$
	Group 2 —	$(t + 0.01) \cdot (w_1 + 0.11 \cdot w_2 + 1.14) \cdot (-0.47)$
Two Tank System sub.	Group 1 —	$\left(-Cv_b \cdot \sqrt{\lvert h_1 - h_2 \rvert} + Q_p\right)/A$
	Group 2 —	Q_p/A
	Group 3 —	$\left(-Cv_b \cdot \sqrt{\lvert h_1 - h_2 \rvert} + Q_p\right)/A$
Two Tank System w/o sub.	Group 1 —	$Q_P/A - \sqrt{Q_P} \cdot h_1$
	Group 2 —	Q_p/A
	Group 3 —	$\sqrt{Cv_b} - A$

for the segmentation and grouping steps of the symbolic regression approach for hybrid automata learning. For most examples, the initial window width l_{init} as well as the length for extending the window l_{step} are relevant for the segmentation while for the grouping especially the relaxation parameter ξ_G for the grouping criterion is of relevance.

Table XIX presents the results of learned expressions for the example systems using the presented approach with the given hyperparameters. An example for the result of the grouping step is illustrated in Figure 67 for the power converter example. The figure shows that the grouping correctly identifies two groups, corresponding to the two

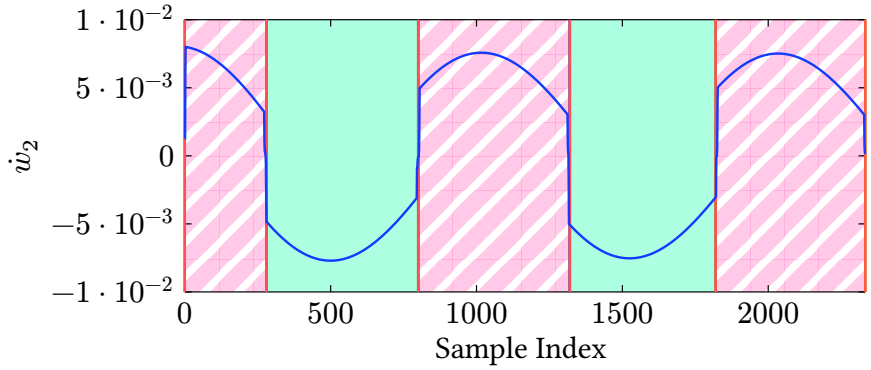


Figure 67 — Segmentation and grouping for the converter example: colors indicate the group of the segment.

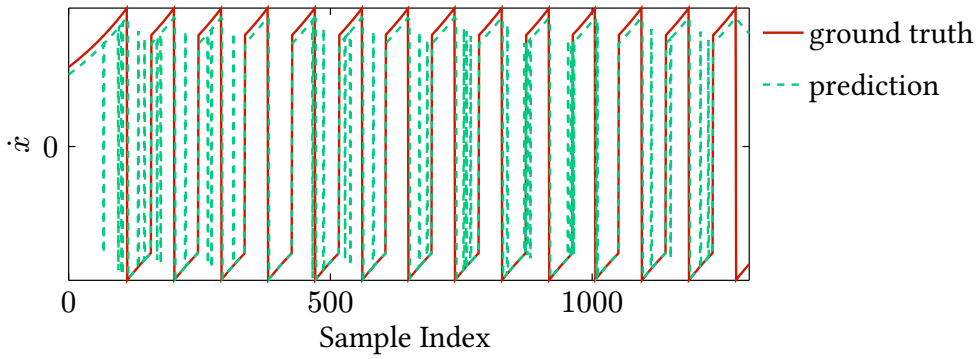
Table XX — Accuracy of the hybrid decision tree in the segmentation step in terms of segmentation metric Ω_{seg}, the grouping step in terms of grouping metric Ω_{group}, and the MSE of the predicted traces on the evaluation set.

Example	Ω_{seg}	Ω_{group}	MSE
Boiler	1.0	$9.50 \cdot 10^{-4}$	$1.26 \cdot 10^{-2}$
Power Converter	2.8	$3.82 \cdot 10^{-7}$	$6.72 \cdot 10^{-7}$
Two Tank System *sub.*	0.0	$3.08 \cdot 10^{-8}$	$2.31 \cdot 10^{-6}$
Two Tank System *w/o sub.*	7.86	$4.93 \cdot 10^{-6}$	$8.58 \cdot 10^{-6}$

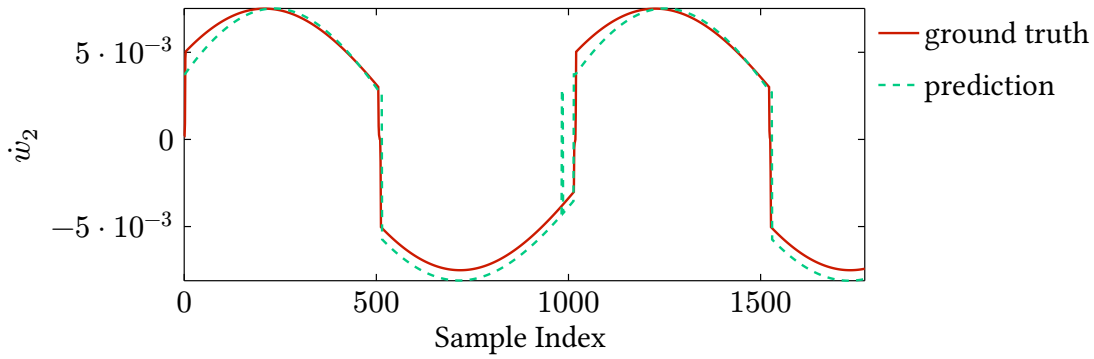
modes of the system. However, some differences remain between the learned expressions and the ground truth, as shown in Table XIX. Notably, in the power converter example, the second learned expression includes the time variable t, even though the true system dynamics do not depend on time. Since the simulation time is very short, the values of t are small, and the learned expression still achieves high accuracy.

The identified expressions for the examples consistently capture at least the most dominant terms of the ground truth expressions, though they may not fully represent less influential components. This is particularly evident in the two tank system: the approach reliably identifies the dominant effect of the pump, represented by the term Q_p/A. However, the influence of water transfer between the tanks is captured precisely only in the scenario where the term $\sqrt{|h_1 - h_2|}$ is explicitly provided as a substitution. In the scenario without this substitution, the method approximates the effect by a linear dependence on h_1, which serves as a reasonable approximation within the relevant variable range. When the substitution is included, both dynamics are accurately recovered. Although three groups are initially formed, the grouping step ultimately recognizes that they represent the same dynamics.

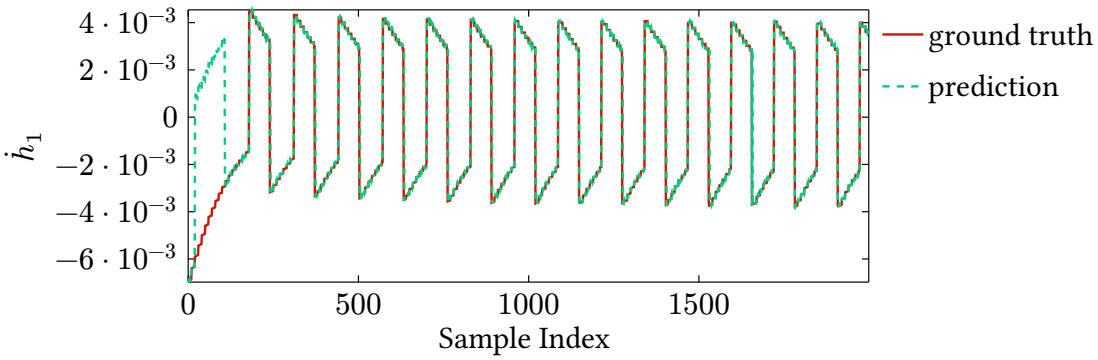
Table XX shows the prediction accuracy of the hybrid decision tree on the evaluation set. The table shows the segmentation objective function Ω_{seg}, the grouping objective



(a) Boiler



(b) Power Converter



(c) Two Tank with substitution

Figure 68 — Predicted (red) and ground truth (blue) evaluation traces.

function Ω_{group}, and the MSE of the predicted traces on the evaluation set. The overall accuracy across all examples is high. The value of Ω_{seg} and Ω_{group} and the MSE for the two tank system with substitution is lower than for the scenario without substitution, demonstrating that incorporating prior knowledge about relevant sub-dynamics enhances the learning process and results in more accurate models. This observation directly supports the previous analysis of the identified expressions. For the power converter a high accuracy is achieved, with low Ω_{seg} and Ω_{group} as well as low MSE.

Figure 68 shows the ground truth and the predicted traces of the evaluation sets. For the boiler, the continuous dynamic behavior is captured well by the model, but

Table XXI — Execution times of the individual steps of model learning with symbolic regression in seconds.

Example	Segmentation Time [s]	Grouping & Characterization Time [s]	Extraction Time [s]
Boiler	2974	791	3
Power Converter	1627	2442	1
Two Tank *sub.*	14800	5576	3
Two Tank *w/o sub.*	2627	6723	3

the decision tree fails to predict the transitions accurately in several cases. For the converter, the traces match well. There is a single miss-prediction of a mode at around sampling index 1000. For all other samples, the mode is predicted correctly by the hybrid decision tree.

For the two tank example, several miss-predictions occur at the beginning of the trace. This is primarily due to differences in the initial values between the evaluation and learning sets. As a result, the initial phase of the evaluation set differs from that of the learning set and can be considered as unseen data during learning. Around sampling index 150, the predicted trace exhibits a discontinuity and subsequently realigns with the ground truth. The remainder of the trace is predicted accurately.

Table XXI presents the execution times of the individual steps of the symbolic regression approach for hybrid decision tree learning. For the two tank system with substitution, the segmentation step is the most time-consuming as the window is extended only by a single sample at a time. Thus, a lot of iterations are needed in the segmentation step to cover the whole trace. In general, the runtime mainly depends on the runtime of SR, which is determined by the number of iterations and the size of the population. The model extraction step is very fast as DTL is used to extract the decision tree from the groups and flow functions, which is a fast and lightweight algorithm. Concluding, our results show that the method accurately identifies the structure of systems from data and predicts the behavior of the system on new data.

6.5 Summary

The results presented in this chapter answer the research question **Q3 — How to learn a model to capture hybrid dynamics of CPSs?** through a systematic investigation of the four sub-questions identified at the beginning of this chapter.

Q3.1 — Which subsequent steps are required for the identification of hybrid automata? We establish a general four-step learning process for hybrid automata identification consisting of trace segmentation, segment grouping, mode characteri-

zation, and model construction (Figure 54). Each step presents distinct algorithmic choices that influence the final model quality and computational efficiency. The steps can be implemented with different paradigms — either focusing on signal similarities (traditional approaches and FaMoS) or actual system dynamics (SR approach) — leading to complementary strengths and application domains.

Q3.2 — How to utilize decision trees for modeling of hybrid dynamics? We introduce hybrid decision trees as an extension of traditional decision trees that captures both discrete transitions and continuous dynamics. Our hybrid decision tree framework provides a foundation for both proposed algorithms. The key innovation lies in incorporating flow functions within decision tree leaf-nodes, enabling the representation of mode-dependent continuous dynamics while maintaining the interpretability and structure of classical decision trees. The decision rules of the hybrid decision tree are designed to capture both the discrete transitions, while every leaf node represents a continuous flow function, i.e., dynamic of the system.

Q3.3 — How can lightweight and intuitive algorithms enable efficient identification of hybrid automata? We achieve with FaMoS computational efficiency through lightweight grouping procedures based on DTW. Our approach prioritizes fast execution and scalability, making it suitable for real-time applications and large datasets. The trade-off lies in focusing on signal similarities rather than actual dynamics, which enables rapid processing but limits the approach to systems with ARX flow functions. For best trade-off, we propose a combination of fast grouping and dynamics-based grouping considering linear differential equations as flow functions.

Q3.4 — How to migrate to a dynamics-based identification using symbolic regression? We present a paradigm shift that integrates the dynamics of the system already in the beginning of the learning process. For this, we use SR in the segmentation and grouping steps. This approach identifies transition points and groups segments based on actual system dynamics rather than signal similarities. Further, SR enables the learning of flow functions without prior knowledge of dynamics types. The SR approach produces compact and meaningful models and introduces new modes only when dynamically necessary. The major advantage lies in the interpretability of the resulting symbolic expressions with the capability to discover the underlying physical relationships governing system behavior.

Overall Assessment: Both proposed approaches successfully enable hybrid automata learning, but address different practical requirements and constraints. FaMoS provides computational efficiency and scalability for systems with known dynamics types, while the SR approach offers flexibility and interpretability for systems with unknown or complex dynamics and focuses on the compactness of models. The choice between approaches depends on the specific application requirements: computational constraints, prior knowledge, and the need for dynamics discovery. Together, these methods establish hybrid automata as viable models for capturing the complexity of CPSs that exhibit both discrete and continuous behavior, completing our progression from purely discrete models through decision trees to hybrid representations.

Chapter 7

Application & Comparison of Learning Approaches

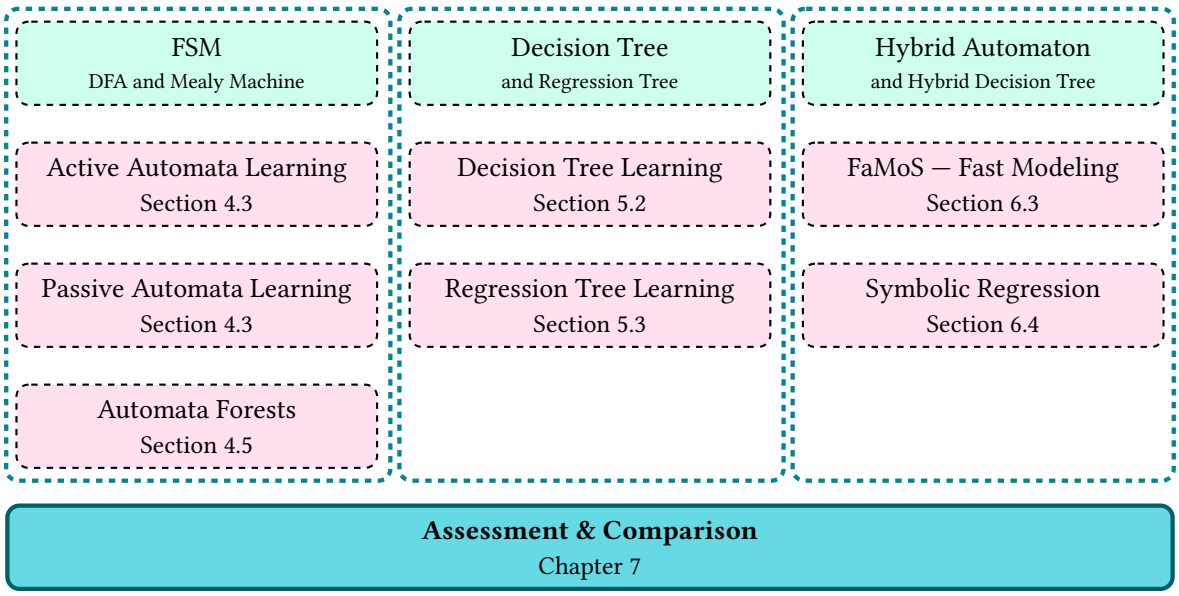


Figure 69 — Overview of the presented models (green) and learning approaches (pink) per model type in this work. The focus of this chapter is the comparison and assessment of the methods.

The diverse modeling approaches presented in the previous chapters — from deterministic finite automata through decision trees to hybrid automata — each address specific

aspects of CPS modeling while introducing distinct capabilities and limitations. As established through our progressive exploration, automata learning provides strong theoretical guarantees and requires deterministic behavior. Decision trees handle non-deterministic, partial, and uninitialized observations but operate on bounded history. Finally, hybrid automata capture continuous dynamics, but introduce computational complexity. The practical application of these modeling strategies to real-world CPSs necessitates a systematic framework for comparing their characteristics and selecting appropriate methods based on application requirements.

Figure 69 shows how the assessment and the comparison of learning approaches build an overarching aspect of this thesis examining the complementarity of different modeling strategies. This chapter synthesizes our investigations by providing both qualitative and quantitative comparisons of the modeling approaches, addressing the fundamental question of when and why to apply each method. Our modeling hierarchy progresses from FSMs — specifically DFAs and Mealy machines for deterministic discrete behavior — through decision trees that accommodate non-deterministic observations, to hybrid automata that integrate continuous dynamics. This progression is further extended through regression trees and hybrid decision trees to encompass both non-deterministic and continuous data characteristics.

We investigate our fourth research question **Q4 — How do different model types and learning strategies for CPSs compare in terms of their characteristics, e.g., interpretability and accuracy?** and the sub-questions:

- Q4.1 — How are learned models integrated in CPS applications?
- Q4.2 — How to systematically compare different modeling strategies?
- Q4.3 — What are the characteristics and trade-offs of the different modeling strategies in this work?
- Q4.4 — How do the models quantitatively compare in terms of their performance metrics?

We begin by examining typical applications that require learned system models, including monitoring, testing, prediction, and explanation. Each application domain imposes distinct requirements on model characteristics, driving the need for diverse modeling approaches. All models presented in this work are designed with interpretability as a primary consideration and typically serve prediction purposes as their core functionality. These predictive capabilities then enable the development of further tools for system monitoring, testing, and diagnosis.

The diversity of modeling challenges in CPS applications necessitates a unified learning framework that accommodates multiple approaches while enabling systematic comparison. We present the Flowcean framework [37], which provides a modular architecture for abstraction, learning, and evaluation of models in CPS-driven applications. The framework has a flexible and extensible design, which allows for the seamless integration of various learning strategies and evaluation metrics, facilitating direct comparison of different approaches.

Motivated by this framework, we establish a comprehensive comparison of the modeling approaches presented in this thesis. Our analysis proceeds in two complementary phases: first, examining qualitative properties of the approaches such as interpretability, robustness, and applicability constraints, followed by quantitative performance assessment using appropriate metrics. Given the fundamental differences between discrete and continuous modeling paradigms, we structure our analysis into separate evaluations for discrete and continuous models to ensure meaningful comparisons. This comprehensive evaluation extends beyond our previously published individual studies by providing an integrated assessment that encompasses both qualitative characteristics and quantitative performance across all modeling approaches presented in this thesis. The results of this evaluation show that through the individual strengths and weaknesses of the modeling approaches, they successfully model a broad range of CPS scenarios and application domains addressing diverse requirements such as interpretability, robustness, and accuracy.

7.1 Related Work

Model learning is applied across various applications and domains, driven by the increasing complexity of modern systems and the need for automated analysis and verification. The landscape of model learning approaches spans from generic machine learning frameworks to specialized tools for CPSs. While the following literature review is not exhaustive, it provides spotlights on developments and approaches in the field.

Generic machine learning frameworks such as scikit-learn [164], PyTorch [165], and TensorFlow [166] offer extensive sets of tools for training machine learning-driven black-box models across diverse applications. While these frameworks provide powerful capabilities for data-driven modeling, they are primarily designed for predictive accuracy rather than interpretability or formal guarantees. In contrast, more specialized approaches have emerged that specifically target the unique characteristics and requirements of CPSs. These frameworks focus on particular aspects such as explaining system behavior [21] or enabling real-time monitoring of CPS [167]. This specialization reflects the recognition that CPS modeling requires distinct considerations beyond traditional machine learning applications, including temporal dynamics and the need for interpretable models.

The broad range of modeling strategies requires a deliberate comparison of their strengths and weaknesses in the context of CPS applications and an informed selection of appropriate techniques for specific use cases. H. Steude et al. [61] presents a comprehensive case study for system identification in CPS, comparing four distinct learning approaches using variational autoencoders, communicating agents, and self-organizing maps. The study evaluates these approaches not only on accuracy metrics but also considers qualitative aspects such as interpretability and applicability to different use-cases. J. Schoukens et al. [13] provides a comprehensive overview of

different approaches for identifying non-linear dynamics, systematically analyzing the individual opportunities and limitations of each method. This work establishes foundations for understanding when and how different identification techniques are effectively applied. Considering these foundations, practical applications of system identification are demonstrated across various domains.

The pursuit of formal guarantees in CPS modeling faces inherent theoretical limitations. R. Alur et al. [168] provides a rigorous analysis of formal properties for hybrid automata, with particular focus on reachability analysis. A key finding of this work is that reachability is undecidable for general multivariate hybrid automata, establishing fundamental computational limits. However, the analysis also identifies that specific restrictions on flow functions restore decidability. E. A. Lee [169] examines additional fundamental constraints, including the treatment of non-deterministic behavior (characterized as 'chaotic' behavior) and the practical challenges of discretizing continuous state spaces for computational simulation. These limitations underscore the importance of carefully considering the trade-offs between model fidelity, computational efficiency, and formal guarantees when selecting modeling approaches for specific applications.

To systematically organize the diverse landscape of learning approaches for CPS, O. Niggemann et al. [73] provides a taxonomy that distinguishes between different types of automaton models. The work uses two key dimensions: the complexity in the number of operational modes and the complexity of the underlying dynamics. This taxonomy categorizes models as memoryless, dynamic, or stochastic, representing, e.g., discrete models, hybrid models, algebraic models, and random models. Such a taxonomy helps to select appropriate modeling approaches based on specific system characteristics and requirements.

The interconnection between discrete and continuous dynamics is a critical aspect of CPS modeling. The theoretical foundations for this integration are explored by M. Bayoudh et al. [170] and H. Zaatiti et al. [16], who discuss the relationship and mapping between discrete automata and hybrid automata or continuous dynamics. These works address how discrete abstractions capture essential properties of continuous systems for diagnosis purposes.

While the literature establishes theoretical foundations and individual approaches, the practical deployment of learned models across diverse CPS applications remains a central challenge.

7.2 Practical & Real-World Applications of Abstract Learned Models

We examine the practical deployment of learned models to understand the specific requirements that guide model selection. Models learned from data serve a variety of purposes, including model-based testing, behavioral prediction, diagnosis, system monitoring, or explanation:

- Monitoring: A model can be used to monitor the system and validate its conformance as demonstrated for decision tree models in Section 5.2.4. High model accuracy is essential to ensure confidence in monitoring results.
- Testing: Test cases are generated from the model of a system. The test cases support the design of the system as well as maintenance and update of the system. In model-based testing, the model is used to generate test cases that cover the functionality of the system. Often, a coverage metric is used to indicate the sufficiency of the generated test cases. Complete coverage is achieved if all functionality according to the coverage criterion has been considered.
- Prediction: Prediction of the system behavior by a learned model is particularly useful for CPSs where the in-field testing is expensive, time-consuming, or restricted by safety regulations. The prediction allows optimizing the system for a given environment.
- Diagnosis: The diagnosis is based on the comparison of the model with the behavior of the system. Additionally, the model might already provide information about the original cause of a deviation from the expected behavior.
- Explanation: An interpretable model explains the behavior of the system. The model must be understandable for the user and provide insights into the system behavior. An explanation model is especially useful if the system is a black-box system, i.e., the user does not have access to the internal behavior of the system. The explanation model potentially improves insights to process requirements, maintenance actions, and maintenance schedules.

With the previous chapters, we cover the applications of learned models in monitoring, testing, and prediction for CPSs as well as the explanation of their behavior. In Chapter 1, we present a case study to explain the behavior of a system and to derive actions to adapt external influences to the requirements of a use-case. This is done in a case study with a Real-Time Localization System (RTLS). We present two methods for test case generation, first, using automata learning from Section 4.4 and second, using decision trees as introduced in Section 5.4. Those case studies use the RTLS and the coffee machine (Section 3.7.1), respectively. Further, we evaluate decision trees and automata in a monitoring scenario as done in Chapter 5. In this scenario, we use the models to predict system behavior, which can be compared against the actual system behavior to identify deviations, which indicate a problem in the system.

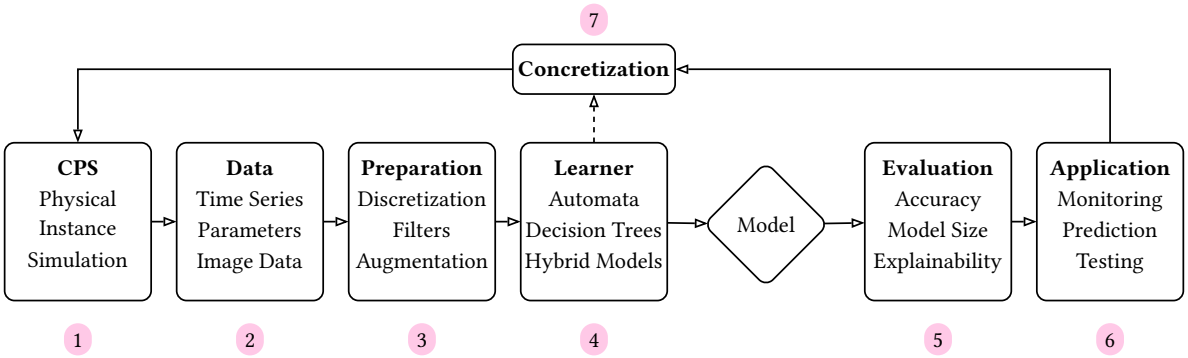


Figure 70 — General learning process for CPS-driven applications consisting of seven steps.

7.3 A General Framework for Learning Models in CPS-driven Applications

We recognize that effective comparison and deployment necessitates a unified framework. The varying demands across monitoring, testing, prediction, and explanation of systems highlight the need for systematic integration and evaluation capabilities that accommodate the distinct characteristics of different modeling strategies.

While the previous chapters demonstrate that learned models successfully drive applications such as test case generation and prediction for CPSs, the heterogeneity of requirements highlights the need for a unified framework that integrates multiple learning strategies while enabling fair comparison.

The diverse learning methods presented in this thesis motivate a framework with universally applicable methods and interoperable solutions for modeling of CPSs from various domains. Despite the diversity in modeling approaches and application scenarios, our case studies from localization systems (as presented in diagnosis and test generation applications) to control systems (as addressed in hybrid automata learning) reveal common patterns in the data-driven modeling process. Even though applications of learned models are diverse, the procedure for learning and applying models follows similar stages that can be systematized and modularized. This general process is shown in Figure 70 and implemented in the framework Flowcean [37].

Starting from a given CPS that can be either a physical instance of a system or a simulation in Step 1, the process receives data from the system in Step 2. As demonstrated throughout this thesis, this data can be highly diverse, ranging from discrete input-output traces for automata learning to continuous time series for hybrid automata identification, as well as configuration parameters and environmental measurements

for decision tree-based diagnosis. The heterogeneous nature of CPS data necessitates a flexible framework that can handle different data characteristics.

Afterward, a preparation in Step 3 performs data preprocessing, e.g., by filtering. This block implements the abstraction strategies, e.g., those discussed in Section 4.2. The abstraction becomes evident when considering the different requirements: automata learning requires deterministic discrete observations, while decision trees can handle non-deterministic behavior, and hybrid automata need to bridge discrete and continuous dynamics. Furthermore, data augmentations might be implemented to improve the learning set for model learning.

In Step 4 of the process, the data is utilized for model learning. The diverse learning strategies developed in this thesis — automata learning (both active and passive), decision tree learning (including regression trees), and hybrid automata identification — address different characteristics of CPS data. Our theoretical analysis has revealed fundamental limitations when learning from bounded history, which motivates the need for multiple complementary approaches within a unified framework. Many of the learners presented in this thesis are part of the Flowcean framework or in the process of being integrated as learner modules.

The learner modules might implement a feedback loop to the system (as shown by the dashed arrows in Figure 70). The feedback loop enables online or active learning strategies, where an interaction or iterative data aggregation from the system is performed. This addresses, e.g., active automata learning from our case studies, which requires direct system interaction. Between the learner and the system, a concretization in Step 7 provides an inverse transformation of the abstraction applied in the preparation step. This concretization is often applied on abstracted representations (e.g., discrete automaton models from continuous systems) that must operate on the concrete system level.

After a model is learned, the evaluation is done in Step 5. Our work has demonstrated that the quality assessment must consider multiple dimensions: prediction accuracy (as analyzed for decision trees), model interpretability (crucial for explanation and diagnosis applications), coverage guarantees (essential for test case generation), and computational efficiency. The evaluation metrics serve to compare different modeling approaches and select the most suitable model for the specific application requirements.

In the final application of Step 6, models are integrated into tools like monitoring, prediction, or testing, which are used with the CPS. The diverse applications presented in this chapter — from diagnosis and explanation to systematic test case generation — demonstrate the practical value of learned models. Concretization in Step 7 is needed in the case of a previous abstraction, ensuring that abstract models can provide actionable insights for real-world CPSs.

Monitoring considers the supervision and detection of faulty or inaccurate system behavior as we have demonstrated with discrete decision tree applications. Testing is the task of evaluating a system with respect to its behavior, which is accomplished

through both automata-based and decision tree-based test generation approaches. Finally, prediction provides information on the behavior of a system in specific scenarios. This is especially useful for the design of new systems and the optimization of system performance as proposed in our diagnosis study of the RTLS.

While existing machine learning frameworks are powerful, they are typically designed for specific learning strategies, such as passive learning on static datasets or reinforcement learning in controlled environments. This specialization limits their flexibility in accommodating different approaches or generalizing across various CPS applications. Our experience with the diverse modeling approaches presented in this thesis — from deterministic automata requiring controlled system access to decision trees learning from non-deterministic observations to hybrid automata bridging discrete and continuous dynamics — highlights the need for a more flexible framework.

The challenges we have identified include:

- *Data heterogeneity*: Different learning approaches require different data formats and preprocessing steps, as evident from our comparison of automata learning (requiring reset-based traces) versus decision tree learning (working with bounded history observations).
- *Abstraction complexity*: Each modeling approach necessitates specific abstraction strategies, from the deterministic discretization required for automata to the continuous-discrete bridging needed for hybrid automata.
- *Evaluation diversity*: Different applications demand different evaluation metrics, from prediction accuracy in monitoring scenarios to coverage guarantees in testing applications.
- *Integration barriers*: Combining multiple modeling approaches or switching between them requires significant reconfiguration and re-implementation effort.

To address these limitations and challenges, the framework Flowcean is designed in a modular manner, offering the following features:

- *Re-usability of learning processes and components:* Flowcean provides a flexible learning pipeline that defines reusable paradigms and interfaces, enabling modules to be applied across different CPS applications without significant reconfiguration.
- *Integration and combination of learning strategies:* Flowcean allows the seamless integration of multiple learning strategies, ranging from passive to active approaches, within a unified framework.

By interfacing with existing learning libraries, Flowcean broadens applicability across diverse domains.

Figure 71 shows the architectural relation between different concepts as well as six steps (A to F) of model generation and evaluation. The modeling task is partitioned into learning and evaluation. The initial step in the workflow involves sourcing data from the system. We refer to a data source as an *environment (A,D)*, which encompasses pre-collected learning sets, online system simulations, or real-world CPSs, providing a generalized interface to both the learning and the evaluation strategy. Data from the environment may undergo pre-processing via *transforms (B)*. Following this, the trans-

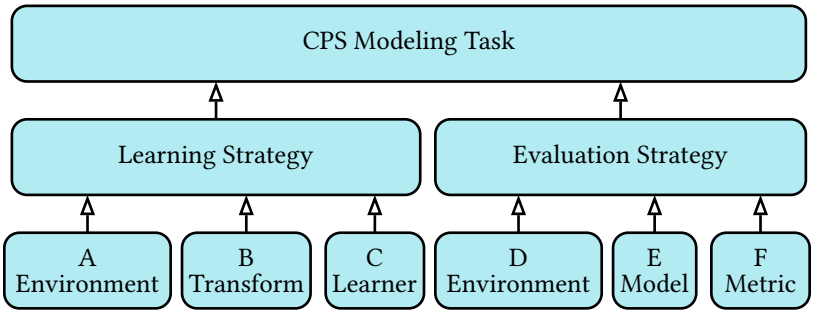


Figure 71 — Component-view of the concepts, showing the three steps of data-driven modeling (A to C) extended by an evaluation (D to F).

formed data is fed to a *learner (C)* to generate a model. The composition of environment, transforms, and the learner constitutes a *learning strategy*. In the *evaluation strategy*, a second environment (D) is used to evaluate the generated *model (E)* on unseen data and to assess its performance using *metrics (F)*.

A key feature of Flowcean is the ability to combine different learning strategies and evaluation metrics in a modular fashion. Flowcean inherently combines three distinct variants of learning strategies: offline, incremental, and active learning. Figure 72 visualizes these learning strategies in a flowchart.

Offline learning strategies involve processing a fixed learning set at once, whereby model parameters are updated collectively as it is done for passive automata learning and decision tree learning in Chapter 4 and Chapter 5, respectively. In this scenario, the learning strategy receives a complete learning set from the environment, which is then used to train the model. An environment provides the data, and the learning strategy selects the respective features for system input and output.

Conversely, incremental learning occurs through continuously updating the model as new data becomes available. That is, an environment incrementally provides data to the learning strategy and in each incremental step, the learner processes small batches of data, potentially even individual samples. The environment autonomously decides which data to present to the learner, rendering the process non-interactive. Learning from streaming data is thus an example of incremental learning. We use streaming data in Section 5.4 when updating current observations of the system.

Active learning engages the learner in actively interacting with the environment. In this scenario, the learning strategy requests an action from the learner, which in turn advances the environment. The environment then provides data that is observed and provided to the learner. We use active automata learning in Chapter 4 and actively interact with a decision tree model in Section 5.4.

An example of a learning pipeline in Flowcean composed of the modules is visualized in Figure 73. The modularization allows running different learning strategies and evaluation metrics on the same data and transforms. Thus, an evaluation of the models is performed in a consistent manner.

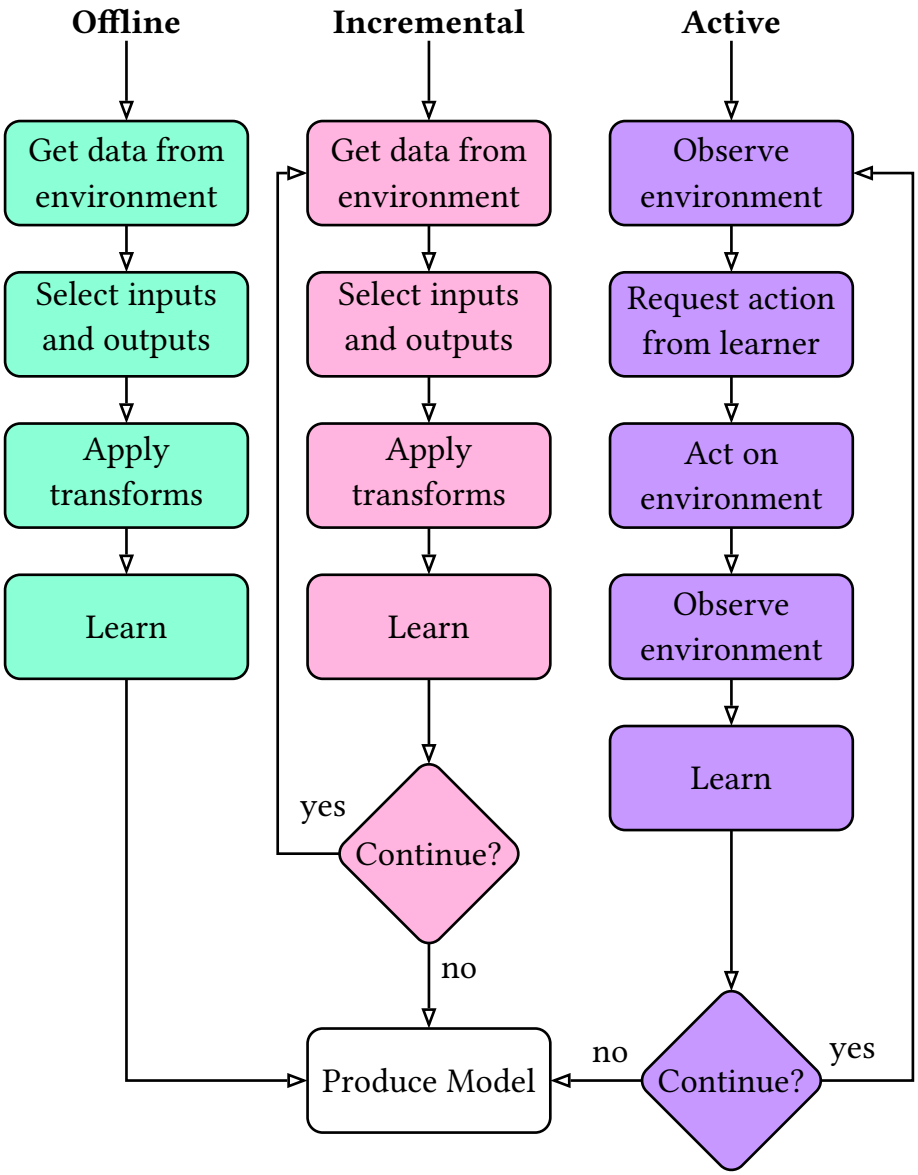


Figure 72 — Flowchart for the learning strategies offline, incremental, and active learning.

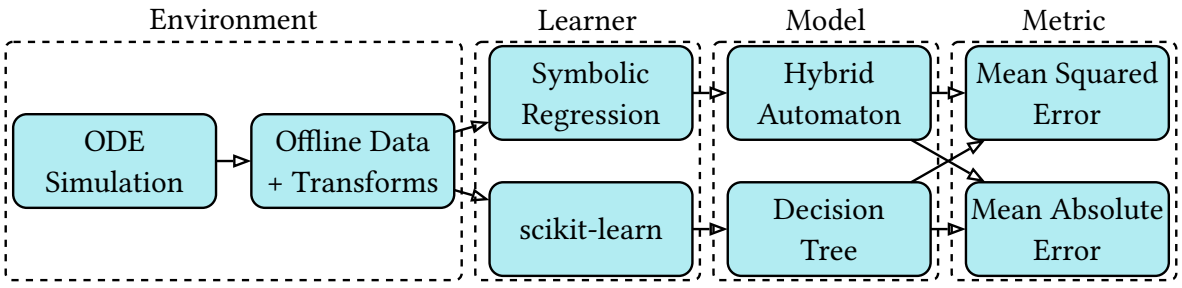


Figure 73 — Illustration of modularization in Flowcean composing the learning and evaluation pipeline.

Table XXII — Properties of the learning approaches in this thesis: Active Automata Learning (AAL), Passive Automata Learning (PAL), Automata Forests (AF), Decision Tree Learning (DTL), Regression Tree Learning (RTL), Fast Modeling for Cyber-Physical Systems (FaMoS), and Symbolic regression for hybrid system identification (SR).

Property	AAL	PAL	AF	DTL	RTL	FaMoS	SR
Model Type	DFA	DFA	DFA	Decision Tree	Regression Tree	Hybrid Decision Tree	Hybrid Decision Tree
Formal Guarantees	Yes	Yes	Partially	Partially	No	No	No
Non-Deterministic Input	No	No	No	Yes	Yes	Yes	Yes
Continuous Input	No	No	No	No	Yes	Yes	Yes
Differential Equations	No	No	No	No	Partially	Yes	Partially
Multi-dimensional Input	Abstracted	Abstracted	Abstracted	Yes	Yes	Yes	No
Infinite Traces	Yes	Yes	Yes	No	No	Yes	Yes
Efficiency Improvements	No	No	Yes	Yes	Yes	Yes	No
Size Improvements	No	No	No	No	Partially	No	Yes

7.4 Model Properties & Capabilities

The Flowcean framework provides the infrastructure for systematic comparison, but effective model selection requires detailed understanding of the individual capabilities and limitations of each approach. We analyze qualitative properties of each learning strategy to facilitate informed decision-making. Motivated by Flowcean, we further perform a quantitative assessment of the learning strategies.

Table XXII provides a comprehensive overview of the characteristics of all learning strategies presented in this work. We consider three model types for CPSs: DFAs, decision and regression trees, and hybrid decision trees. For DFAs, we employ three learning strategies: passive automata learning (PAL), active automata learning (AAL), and automata forests (AF). We consider the RPNI and the L*-algorithm as introduced in Chapter 3, which learn deterministic DFAs and Mealy machines. We use DTL and RTL for decision and regression trees, respectively. For hybrid decision trees, we present two identification algorithms: FaMoS and an approach using SR.

The progression from discrete to continuous models reveals distinct properties and capabilities. Automata models provide formal guarantees through their deterministic structure and proven equivalence to target systems. Decision tree models preserve spe-

cific properties: they over-approximate the behavior of underlying DFAs (as proven in our decision tree analysis), maintaining consistency while trading formal completeness for practical flexibility. Automata forests stochastically retain consistency properties of DFAs on the learning set, while requiring characteristic training data for finding exact models. Continuous learners (RTL, FaMoS, and SR) operate on infinite input spaces where formal guarantees cannot be established through traditional automata-theoretic methods. Instead, we provide empirical validation through accuracy metrics.

The progression from discrete to continuous models enables handling increasingly complex data characteristics. DFAs require deterministic input sequences, while DTL, RTL, FaMoS, and SR-based identification process non-deterministic data, and RTL, FaMoS, and SR-based identification handle continuous inputs. The progression to continuous models comes at the cost of formal guarantees: while DFAs provide strong theoretical foundations, continuous models prioritize empirical performance and adaptability. Randomized learning strategies such as SR cannot guarantee convergence to a target model, but approximate complex dynamics effectively. RTL, FaMoS, and SR-based identification potentially identify differential equation: RTL and the SR-based approach can provide predictions for derivatives, while FaMoS directly predicts system evolution.

DTL and RTL support multi-dimensional input through feature vectors, and FaMoS learns multi-dimensional differential equations. The SR-based identification learns individual symbolic expressions per mode, currently limited to single-dimensional dynamics. DFAs operate on single-dimensional deterministic sequences, requiring abstraction layers for multi-dimensional input as demonstrated in our abstraction strategies in Section 4.2.

Temporal modeling strategies distinguish the approaches fundamentally. All learning approaches for DFAs learn from infinite traces, capturing unbounded temporal state transitions. DTL and RTL operate on bounded history observations, enabling resource-efficient practical deployment while limiting temporal expressiveness. Identification methods for hybrid decision trees utilize continuous traces, which are segmented into intervals capturing the dynamic modes of the system.

We implement optimizations for practical deployment and model efficiency of learning strategies. Decision tree models require unpruned structures to maintain formal guarantees as proven in our theoretical analysis of Chapter 5, while regression trees apply pruning techniques to reduce model size and to avoid overfitting. Automata forests improve accuracy on smaller, non-characteristic learning sets compared to classical automata learning, addressing practical data limitations. Our hybrid decision tree approaches provide orthogonal improvements: FaMoS optimizes runtime efficiency through fast segmentation and grouping of traces, while the SR-based approach reduces model complexity through dynamics-driven identification of symbolic expressions.

Beyond the explicit properties in Table XXII, interpretability, expressiveness, and robustness characterize the practical impact of our models. All presented learning

approaches target inherently interpretable model types. Decision trees provide an intuitive structure and visualization capabilities, where our decision tree models abstract temporal behavior into feature representations. Automata models offer inherent expressiveness for complex temporal behaviors. Hybrid automata extend this expressiveness with representations for continuous dynamics. FaMoS handles ARX model dynamics, while the SR-based approach provides flexibility across different types of dynamics.

Robustness characteristics vary across the approaches. Hybrid methods depend on hyperparameter selection, potentially limiting robustness. DTL and RTL may overfit training data, but the approximation is estimated through leaf node analysis as demonstrated in Chapter 5. Learning methods of DFAs are robust on characteristic training data, where they consistently identify the target model. However, a characteristic dataset cannot be guaranteed for black-box systems. Automata forests address this limitation by employing multiple models to capture diverse system aspects — improving robustness and accuracy at the cost of increased model complexity.

Summarizing, all approaches demonstrate complementary strengths and targeted trade-offs for different requirements. Building on this qualitative analysis, we turn at the quantitative performance evaluation. The quantitative analysis validates the practical effectiveness of each approach and showcases individual strengths and weaknesses.

7.5 Model Performance

In the previous chapters, we evaluate each learning approach on representative examples to demonstrate their practical effectiveness and performance. Table XXIII summarizes the evaluation coverage across all approaches and the examples from Section 3.7. A '✓' denotes full evaluation of the approach, while a '+' indicates evaluation of a variant or modified version. For example, the evaluation of passive automata learning is done as part of the decision tree evaluation using bounded rather than infinite observations. The evaluation of DTL and RTL on the RTLS demonstrates the effectiveness of the approaches in the motivating learning scenario from Chapter 1.

Each approach employs domain-appropriate abstractions and evaluation metrics:

- **Automata learning** (active and passive): discrete abstraction, evaluated by the number of queries and data needed for accurate modeling,
- **Automata forests**: discrete abstraction, evaluated by the number of correctly translated input traces,
- **Decision trees**: discrete abstraction, assessed through accuracy, true positive, and false positive metrics,
- **Regression trees**: continuous abstraction, evaluated by mean prediction error,
- **Hybrid decision trees** (FaMoS and SR-based): continuous abstraction, measured by MSE.

We execute a qualitative comparison for the heterogeneous evaluation scenarios through color-coded performance indicators in Table XXIII. Active and passive

Table XXIII — Usage of examples in the evaluation of the different learning approaches: Active Automata Learning (AAL), Passive Automata Learning (PAL), Automata Forests (AF), Decision Tree Learning (DTL), Regression Tree Learning (RTL), Fast Modeling for Cyber-Physical Systems (FaMoS), and Symbolic regression for hybrid system identification (SR). A '✓' indicates that the approach is evaluated with the example. A '+' indicates that a variant of the approach is evaluated on the example. The color coding shows the performance of the model, where a green cell indicates good performance, yellow indicates moderate performance, and orange indicates poor performance.

Example	AAL	PAL	AF	DTL	RTL	FaMoS	SR
Coffee Machine		+	✓	✓			
Water Tank		+		✓	✓		
Two Tank						✓	✓
Boiler		+	✓	✓	✓	✓	✓
Power Converter						✓	✓
Vacuum Cleaner	✓	✓					
RTLS		✓		+	+		
Artificial			✓	✓			

automata learning demonstrate effectiveness on practical systems, producing interpretable and compact models for the RTLS and vacuum cleaner examples. Passive automata learning achieves exact model identification for the coffee machine under limited data, but exhibits reduced accuracy (sub-100% true positive rates) for other systems like the water tank and the boiler. Automata forests consistently outperform classical automata learning across all evaluated examples when learning from limited datasets, validating their robustness to incomplete training data.

Decision tree analysis of the RTLS in Chapter 1 provides accurate behavioral models only with constrained resolution in the clustering of data. Historical data incorporation in our decision tree models in Chapter 5 significantly improves generalization and accuracy, which is shown in the results for the water tank and boiler systems. For systems exhibiting observation ambiguity (coffee machine and artificial examples), decision trees face fundamental limitations proven in Chapter 5, but provide valuable uncertainty quantification through impure leaf nodes.

The regression tree of the RTLS in Chapter 1 exhibits overfitting compared to the discrete decision tree approach, demonstrating the trade-offs between continuous modeling and generalization. Historical data integration in Chapter 5 substantially improves accuracy for the water tank and boiler examples, highlighting the importance of temporal context.

FaMoS demonstrates good performance on linear systems (boiler and power converter) but shows limitations for non-linear dynamics (two-tank system). The SR-based

approach effectively captures complex non-linear dynamics, while maintaining good performance on the two-tank system and the boiler.

As introduced in the preliminaries, all example systems are presented as components of a coffee machine. This modular perspective enables the integration of models for individual subsystems, such as the water tank, boiler, and power converter, to construct comprehensive representations of complex systems. By combining the learned models, we achieve accurate and interpretable models for complex systems through systematic composition of their constituent elements and different levels of system abstraction.

The evaluation demonstrates that every example system achieves satisfactory performance with at least one learning approach, validating the complementary nature and broad applicability of our modeling strategies. This coverage confirms that our approaches address diverse CPS characteristics through appropriate selection of learning methods.

Cross-approach quantitative comparison requires domain-specific evaluation metrics due to fundamental differences between discrete and continuous modeling paradigms. In the following, we organize quantitative comparisons within homogeneous model categories, enabling performance assessment of our approaches.

7.5.1 Quantitative Comparison of Discrete Models

Setup

Examples:

- Coffee Machine: coffee machine from Section 3.7.1
- Water Tank: flushing variant of the water tank from Section 3.7.2
- Boiler: boiler system from Section 3.7.4 in the controlled temperature range variant. The output signal, i.e., the temperature of the boiler, is discretized to a set of three values representing the intervals $[0, 20]$, $(20, 23]$, and $(23, 26]$.

Language:

- Automata: Java
- Automata Forest: Java
- Decision Tree: Matlab

Scenarios:

- coffee machine: discrete system representation
- water tank: sampling period $t_S = 0.1$
- boiler: sampling period $t_S = 10$

Learning Set: 200 random sequences of length between 10 and 50 starting in the initial state

- for DTL, all subsequences of length n are extracted from the sequences
- for automata learning, the sequences are used as they are

- automata forests use the same learning set as automata learning and the number of automata in the forest is $m = 50$ and the learning set ratio per instance is $\rho = 0.8$.

Test Set: 50 traces
Metrics: accuracy, i.e., the ratio of correctly predicted instances to the total instances.
Device: IntelCore i7@2.90GHz, 32GB RAM

We integrate the findings and evaluations of Chapter 4 and Chapter 5 into a quantitative comparison of discrete models.

As discussed in Chapter 5, a comparison of decision tree models and automata models including automata forests, requires a pre-processing of data. The automaton and automata forest models are trained directly on sequences starting in the initial state, while for decision trees, we extract all subsequences of bounded length n from these sequences to form the learning set. As a result, the automaton model benefits from access to longer, uninterrupted sequences, potentially providing more information than the bounded traces.

Consequently, the evaluation of the automaton models and decision tree is performed on the same test set as in Chapter 5. We evaluate the automaton model on the same test set of observations of bounded history n as the decision tree. We use the automaton model to search for the occurrence of the last $n - 1$ samples in a path of the automaton. The next output is then predicted based on the transition for the next input symbol that follows this path in the automaton. For the automata forest, we use the same approach and apply the majority vote variant, which shows best performance in Section 4.5. Thus, we perform the prediction for every instance in the forest individually and take the majority vote as the final prediction.

The results of the quantitative comparison of discrete models are shown in Table XXIV. We observe that the different modeling approaches exhibit varying levels of accuracy across the different examples. For the coffee machine, the automaton model achieves perfect accuracy of 100% due to the deterministic nature of the system and the availability of characteristic training data. For the other examples, the automaton model struggles to achieve similar levels of accuracy, indicating potential limitations in capturing the underlying dynamics of these systems. A reason for this discrepancy is the fixed learning set size, which does not necessarily reflect the characteristics of the system. Further, the evaluation scenario is to the detriment of the automaton as it uses the first occurrence of a sequence to predict the next output.

The automata forest benefits from the ensemble approach, allowing to capture a wider range of behavior and, thus, consistently showing improved performance compared to traditional automata learning, e.g., for the boiler example, where the single automaton struggles to generalize. The learning time of the automata forests is significantly longer than that of the single automaton due to the need to train multiple

Table XXIV — Accuracy (Acc.) in % and learning time t_{learn} in seconds for the discrete models, learned with passive automata learning, automata forests and DTL.

Example	n	Automaton		Automata Forest		Decision Tree	
		Acc.	t_{learn} [s]	Acc.	t_{learn} [s]	Acc.	t_{learn} [s]
Coffee Machine	6	100.00	0.06	100.00	0.4	99.92	0.02
Water Tank	4	93.96	0.14	93.96	0.9	93.96	0.15
Boiler	2	49.88	1.07	94.72	30.7	95.46	1.40

models and aggregate their predictions. The learning time for single automata and decision trees is very short and in a similar range.

Table XXIV shows that with smaller history, i.e., smaller n, the decision tree model achieves better accuracy. Thus, we conclude that the decision tree model is more effective in scenarios with limited temporal context, while the automaton model is better suited for scenarios with longer temporal dependencies. Automata learning identifies a ground truth model precisely under sufficient learning data. Meanwhile, the decision tree and the automata forests are more robust to limited learning sets.

7.5.2 Quantitative Comparison of Continuous Models

Setup

Examples:

- Boiler: the controlled temperature range scenario of the boiler example from Section 3.7.4
- Power Converter: generic variant of the power converter example from Section 3.7.5.
- Two Tank: two tank system example from Section 3.7.3.

Language:

- FaMoS, Regression Tree: Matlab,
- Hybrid Decision Tree with SR: Python

Learning Set:

- FaMoS on the boiler example: 8 traces
- others: 1 trace per example

Test Set: 1 trace per example

Metrics: MSE: Mean-square error of the predicted trace to the ground truth trace of the derivative of the variables, i.e.,

- derivative of the temperature for the boiler example
- derivative of the output voltage for the power converter example
- derivative of the level of the first tank for the two tank example

Device: IntelCore i7@2.90GHz, 32GB RAM

Table XXV — Mean square error (MSE) of the continuous approaches on the evaluation trace of the examples and learning time t_{learn} in seconds.

Example	Regression Tree		Hybrid DT FaMoS		Hybrid DT Symbolic	
	MSE	t_{learn} [s]	MSE	t_{learn} [s]	MSE	t_{learn} [s]
Two Tank	$3.56 \cdot 10^{-7}$	1	$1.38 \cdot 10^{-7}$	1	$8.58 \cdot 10^{-6}$	9353
Power Converter	$2.38 \cdot 10^{-7}$	< 1	$3.97 \cdot 10^{-6}$	6	$6.72 \cdot 10^{-7}$	4070
Boiler	$1.02 \cdot 10^{-2}$	12	$1.39 \cdot 10^{-2}$	17	$1.26 \cdot 10^{-2}$	3768

From the findings of Chapter 5 and Chapter 6, we perform a quantitative analysis and comparison of the learning approaches on continuous data presented in this work. We compare regression tree models and hybrid decision trees (FaMoS and based on SR) in terms of their performance metrics.

The direct comparison of the three approaches is challenging due to their differing objectives and output representations. FaMoS learns difference equations for direct variable prediction, enabling immediate system state forecasting. In contrast, the SR-based approach and regression trees provide predictions for derivatives of the system variables without directly modeling the variables themselves. Thus, for a comparative analysis, we focus on the accuracy of predictions for derivatives across all three approaches.

The results in Table XXV reveal performance characteristics across the three continuous modeling approaches. Overall, the accuracy of all learners across the examples is high and in a similar range. For the two-tank system, FaMoS achieves the highest accuracy followed by regression trees and the symbolic regression approach. For the power converter and the boiler, the symbolic regression approach shows slightly better performance than FaMoS. The regression tree has the best accuracy for the boiler and the power converter. Nevertheless, the interpretability of the hybrid decision trees remains a significant advantage as they provide structured insights in the dynamic flow functions of the systems. The learning time of the different approaches varies, with RTL requiring the least amount of time for training, followed by FaMoS. The symbolic regression approach has the longest learning time on the presented examples because of the iterative update of learned expressions.

This analysis shows that no single continuous modeling approach dominates across all examples. All approaches show high accuracy and robustness, while none of them consistently outperforms the others. At the same time, the approaches exhibit distinct qualitative characteristics, particularly in terms of interpretability and the types of dynamics modeled.

All model types of this thesis (discrete and continuous models) are evaluated on the boiler example, which has both discrete and continuous representations. This

allows for a direct comparison of the performance in capturing the system behavior over all approaches. For the discrete approaches best performance is achieved with the decision tree model and for the continuous approaches best performance in the evaluation is with the regression tree model. Nevertheless, all models except for classical automata learning exhibit good performance on the boiler example. Thus, the presented approaches demonstrate a relevant impact in the field of model identification for CPSs.

7.6 Summary

This chapter provides a comprehensive comparison of the diverse modeling approaches presented in this thesis, addressing our fourth research question **Q4 — How do different model types and learning strategies for CPSs compare in terms of their characteristics, e.g., interpretability and accuracy?** through systematic analysis of both qualitative properties and quantitative performance.

Q4.1 — How are learned models integrated in CPS applications?

Our analysis reveals that learned models serve several application domains in CPS contexts.

- In monitoring applications, models validate system conformance by comparing predicted behavior against actual observations, with decision tree models demonstrating particular effectiveness for anomaly detection.
- For testing, both automata-based and decision tree-based approaches enable systematic test case generation, with automata providing comprehensive coverage metrics and decision trees offering guided testing without requiring system resets.
- Prediction applications leverage the learned models to forecast system behavior in scenarios where in-field testing is expensive or safety-critical, particularly valuable for optimization and design purposes.
- Diagnosis applications utilize model-data comparisons to identify deviations and potential root causes, with interpretable models providing insights into system failure modes.
- Finally, explanation applications exploit the inherent interpretability of modeling approaches to provide understandable insights into black-box system behavior, supporting, e.g., maintenance decisions and process optimization.

Q4.2 — How to systematically compare different modeling strategies?

We address the challenge of comparing various types of modeling strategies through the introduction of the Flowcean framework, which provides a modular architecture for abstraction, learning, and evaluation of models in CPS-driven applications. The framework enables systematic comparison by standardizing the learning pipeline. This modular design accommodates three distinct learning strategies (offline, incremental, and active learning) while providing consistent evaluation metrics across different modeling approaches. The flexibility of the framework allows direct comparison of

approaches with different characteristics, from deterministic automata requiring controlled system access to decision trees learning from non-deterministic observations to hybrid automata bridging discrete and continuous dynamics.

Q4.3 — What are the characteristics and trade-offs of the different modeling strategies in this work?

The modeling approaches demonstrate complementary strengths and distinct trade-offs across multiple dimensions.

- Formal guarantees are strongest in automata models, which provide proven equivalence to target systems, while decision trees offer over-approximation properties and continuous models rely on empirical validation.
- Data handling capabilities show progression in the proposed methods: automata require deterministic input sequences, decision trees accommodate non-deterministic observations but operate on bounded history, and hybrid approaches handle continuous dynamics.
- Temporal expressiveness distinguishes the approaches, with automata capturing unbounded temporal dependencies, decision trees providing bounded history modeling, and hybrid approaches segmenting continuous traces into discrete intervals.
- Interpretability remains high across all approaches, though with different characteristics: automata provide intuitive state-transition structure, decision trees offer transparent rule-based decisions, and hybrid automata combine discrete logic with continuous dynamic representation.
- Robustness varies significantly, with automata requiring characteristic training data for deterministic performance, decision trees demonstrating flexibility with incomplete datasets, and hybrid methods depending on hyperparameter selection.

Q4.4 — How do the models quantitatively compare in terms of their performance metrics?

Our systematic comparison methodology separates discrete and continuous model evaluation to ensure meaningful performance assessment despite the fundamental differences in modeling paradigms. The evaluation across representative CPS examples reveals that every system achieves satisfactory performance with at least one learning approach, validating the complementary nature of our modeling strategies. For discrete models, automata achieve perfect accuracy on deterministic systems with characteristic training data, while decision trees demonstrate superior robustness to limited datasets and bounded history constraints. Automata forests consistently outperform classical automata learning across all evaluated examples when learning from limited datasets, addressing practical data limitations in real-world applications. For continuous models, regression trees, FaMoS, and the symbolic regression approach effectively capture complex, potentially nonlinear dynamics and achieve high performance. While regression trees show slightly better performance in two out of three examples, the hybrid decision trees provide explicit expressions for the underlying dynamics of the

systems. The evaluation confirms that appropriate method selection based on system characteristics and application requirements enables effective modeling across diverse CPS scenarios.

The comprehensive comparison establishes guidelines for model selection: choose automata learning for deterministic systems requiring formal guarantees, decision trees for non-deterministic systems or practical limitations, e.g., observation with bounded history and limited data, and hybrid approaches for systems with significant continuous dynamics. The Flowcean framework enables practical deployment of this selection strategy through systematic evaluation and comparison capabilities, providing a foundation for informed modeling decisions in CPS applications.

Chapter 8

Conclusion

Data-driven model learning represents a transformative approach for the analysis and design of CPSs. This work establishes a comprehensive framework of novel learning methods that spans the complete spectrum from discrete to hybrid models, fundamentally advancing the state-of-the-art in CPS modeling. Through algorithmic innovation and rigorous theoretical foundations, we introduce concepts that overcome critical limitations of existing approaches such as the reliance on deterministic abstractions in traditional automata learning and the challenges of modeling continuous dynamics, which we address through hybrid automata learning. Application scenarios demonstrate not only the broad applicability, but also the practical progression and impact of the proposed methods across diverse domains. A comprehensive evaluation of the proposed methods highlights their strengths and weaknesses, which showcases the complementarity of the approaches and provides valuable insights for future research directions.

8.1 Impact

Returning to the motivating example introduced at the beginning of this thesis, the challenges and opportunities of data-driven model learning for complex systems become tangible. The methods and insights developed throughout this work directly address the core issues highlighted in that example, such as the need for models that capture temporal as well as mixed discrete-continuous dynamics. The assessment of our methods demonstrates that interpretable and robust models are identified in the presence of uncertainty and hybrid dynamics. This connection underscores the practical relevance and applicability of the proposed approaches.

This work delivers three major contributions that advance the field: 1) automata learning techniques specifically tailored for CPS environments, 2) novel DTL algorithms with formal learnability analysis that establish theoretical foundations, and 3) pioneering algorithms for hybrid automata learning that bridge the discrete-continuous modeling gap. All these contributions derive or contribute to the generation of interpretable models that are essential for understanding the behavior of CPSs and ensuring trustworthiness and reliability of systems.

Our advances to grammatical inference establish a new paradigm for learning interpretable and highly accurate models of CPSs. Recognizing the limitations of traditional automata learning — particularly its dependency on input data, which cover the entire system behavior — we develop automata forests, an extension that enhances robustness under real-world uncertainty conditions. This advancement represents a shift to robust, uncertainty-aware learning algorithms inspired by the principles of ensemble learning.

The inherent limitations of deterministic abstractions in practical CPS modeling necessitated a rethinking of the learning paradigm. We propose sophisticated DTL algorithms focusing CPS modeling. These algorithms achieve high accuracy while maintaining interpretability. Furthermore, our theoretical analysis reveals the connection between our decision tree models and DFAs. This analysis identifies fundamental limitations of model learning under finite time horizons.

Addressing the challenge of modeling continuous dynamics within CPSs, this work introduces two data-driven learning algorithms for hybrid automata. Our first algorithm employs sophisticated heuristics that achieve efficiency while maintaining intuitive interpretability. The second algorithm establishes a new learning paradigm by exploiting system dynamics to automatically identify transition points between dynamic modes. Through integration of symbolic regression for continuous dynamics identification, our method achieves seamless coupling between discrete and continuous system components.

The impact of our contributions is demonstrated through comprehensive real-world applications. Our diagnostic capabilities enable insights for system integration optimization. The automated test case generation framework, enhanced by our novel test coverage measure for decision trees, establishes robustness validation for CPSs.

A comprehensive comparison demonstrates the versatility of the approaches covering various modeling properties such as interpretability, robustness, and adaptability. Several examples from diverse domains validate the performance and applicability of the proposed methods. In addition, a quantitative analysis of the results highlights the trade-offs between different modeling strategies and showcases the advancement to the state-of-the-art in the identification of models for CPSs.

The theoretical insights and practical implementations presented in this work collectively constitute an advancement that build a base for future direction of CPS research and development.

8.2 Future Work

Flowcean represents a foundational framework for data-driven CPS modeling. The strategic integration of advanced tools from across the CPS applications amplifies the impact and accessibility of our learning methods, enabling widespread adoption across diverse domains and industrial applications.

A particularly promising future application lies in the development of explanation modules that enable us to understand and interact with CPSs. These modules reason about system failures and anomalies detected during monitoring, transforming reactive maintenance into proactive system optimization. By leveraging the inherent expressiveness of abstract models, explanations provide insights into system behavior and failure causation. New methods are needed to extract explanations from abstract models, enabling users to understand the underlying system behavior and the reasons for specific decisions made by the model. The integration of such explanation capabilities establish model interpretability and trustworthiness in critical system applications.

The integration of prior knowledge into the learning process represents another promising avenue for future research. Domain-knowledge not only enhances the learning process, but also facilitates the identification of hyperparameters and the refinement and update of models. Developing methods for model refinement offers new opportunities to reuse existing models, rather than training from scratch, and reduce the need for new data [171].

The domain of hybrid automata learning emerges as a significant opportunity for future advances. Building upon our symbolic regression approach for hybrid system identification, future developments should establish comprehensive learning strategies that seamlessly integrate identified continuous dynamics in the further learning process. This integration enables adaptive model complexity based on system requirements, where transitions to new dynamics occur only when existing representations prove insufficient [172]. This perspective challenges the rigid step-by-step pipeline for the identification of hybrid automata and, instead, requires a flexible learning pipeline with feedback loops between the stages. The ultimate vision is a unified global approach to hybrid automata learning that dynamically balances model complexity, continuous dynamics representation, and system behavioral fidelity, establishing the foundation for autonomous CPS modeling and optimization.

Bibliography

[1] G. D. Putnik, L. Ferreira, N. Lopes, and Z. Putnik, "What is a cyber-physical system: Definitions and models spectrum," *FME Transactions*, vol. 47, no. 4, 2019, doi: 10.5937/fmet1904663P.

[2] R. V. Yohanandhan, R. M. Elavarasan, P. Manoharan, and L. Mihet-Popa, "Cyber-physical power system (CPPS): A review on modeling, simulation, and analysis with cyber security applications," *IEEE Access*, vol. 8, pp. 151019–151064, 2020, doi: 10.1109/ACCESS.2020.3016826.

[3] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: Algorithmic verification and debugging," *Communications of the ACM*, vol. 52, no. 11, pp. 74–84, Nov. 2009, doi: 10.1145/1592761.1592781.

[4] S. Mi, Y. Feng, H. Zheng, Z. Li, Y. Gao, and J. Tan, "Integrated intelligent green scheduling of predictive maintenance for complex equipment based on information services," *IEEE Access*, vol. 8, pp. 45797–45812, 2020, doi: 10.1109/ACCESS.2020.2977667.

[5] M. M. Kumbure, C. Lohrmann, P. Luukka, and J. Porras, "Machine learning techniques and data for stock market forecasting: A literature review," *Expert Systems with Applications*, vol. 197, p. 116659, 2022, doi: 10.1016/j.eswa.2022.116659.

[6] L. Ljung, *System identification: Theory for the user*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[7] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. MIT press, 2017.

[8] T. Sanislav and L. Miclea, "Cyber-physical systems-concept, challenges and research areas," *Journal of Control Engineering and Applied Informatics*, vol. 14, no. 2, pp. 28–33, 2012.

[9] B. Aichernig, R. Bloem, M. Ebrahimi, M. Horn, F. Pernkopf, W. Roth, A. Leitner, M. Tappler, and M. Tranninger, "Learning a behavior model of hybrid systems through combining Model-based testing and machine learning," in *Testing Software and Systems (TSS)*, Springer, 2019, pp. 3–21. doi: 10.1007/978-3-030-31280-0_1.

[10] F. Howar, B. Steffen, and M. Merten, "Automata learning with automated alphabet abstraction refinement," in *International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2011, pp. 263–277. doi: 10.1007/978-3-642-18275-4_19.

[11] E. M. Clarke, "Model checking," in *Foundations of Software Technology and Theoretical Computer Science*, S. Ramesh and G. Sivakumar, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 54–56. doi: 10.1007/BFb005802.

[12] M. S. Branicky, "Introduction to hybrid systems," in *Handbook of Networked and Embedded Control Systems*, Boston, MA: Birkhäuser Boston, 2005, pp. 91–116. doi: 10.1007/0-8176-4404-0_5.

[13] J. Schoukens and L. Ljung, "Nonlinear system identification: A User-oriented road map," *IEEE Control Systems Magazine*, vol. 39, no. 6, pp. 28–99, 2019, doi: 10.1109/MCS.2019.2938121.

[14] J. L. Piovesan, H. G. Tanner, and C. T. Abdallah, "Discrete asymptotic abstractions of hybrid systems," 2006, pp. 917–922. doi: 10.1109/cdc.2006.377733.

[15] G. Sciavicco and I. E. Stan, "Knowledge extraction with interval temporal logic decision trees," in *International Symposium on Temporal Representation and Reasoning (TIME)*, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, pp. 9:1–9:16. doi: 10.4230/LIPIcs.TIME.2020.9.

[16] H. Zaatiti, L. Ye, P. Dague, J.-P. Gallois, and L. Travé-Massuyès, "Abstractions refinement for hybrid systems diagnosability analysis," in *Diagnosability, Security and Safety of Hybrid Dynamic and Cyber-physical Systems*, M. Sayed-Mouchaweh, Ed., Cham: Springer International Publishing, 2018, pp. 279–318. doi: 10.1007/978-3-319-74962-4_11.

[17] J. N. Kapur, *Mathematical modelling*. New York.: Wiley, 1988. doi: 10.1515/9781683928737.

[18] F. E. Cellier, *Continuous system modeling*, 2nd ed. New York: Springer, 1991. doi: 10.1007/978-1-4757-3922-0.

[19] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019, doi: 10.1038/s42256-019-0048-x.

[20] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and regression trees*. in The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984. doi: 10.1201/9781315139470.

[21] M. Blumreiter, J. Greenyer, F. J. Chiyah Garcia, V. Klös, M. Schwammberger, C. Sommer, A. Vogelsang, and A. Wortmann, "Towards Self-explainable cyber-physical systems," in *International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 543–548. doi: 10.1109/MODELS-C.2019.00084.

[22] G. Saveri and L. Bortolussi, "Retrieval-augmented mining of temporal logic specifications from data," in *Machine Learning and Knowledge Discovery in Databases*, 2024, pp. 315–331. doi: 10.1007/978-3-031-70368-3_19.

[23] D. Della Monica, G. Pagliarini, G. Sciavicco, and I. E. Stan, "Decision trees with a modal flavor," in *Advances in Artificial Intelligence (AIxIA)*, A. Dovier, A. Montanari, and A. Orlandini, Eds., Berlin, Heidelberg: Springer-Verlag, 2023, pp. 47–59. doi: 10.1007/978-3-031-27181-6_4.

[24] A. Vignolles, E. Chanthery, and P. Ribot, "Hybrid model learning for system health monitoring," in *Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS)*, Pafos: Science Direct, 2022, pp. 7–14. doi: 10.1016/j.ifacol.2022.07.098.

[25] S. Plambeck, G. Fey, J. Schyga, J. Hinckeldeyn, and J. Kreutzfeldt, "Explaining cyber-physical systems using decision trees," in *International Workshop on Computation-aware Algorithmic*

Design for Cyber-physical Systems (CAADCPS), 2022, pp. 3–8. doi: 10.1109/CAADCPS56132.2022.00006.

[26] J. Leonard and H. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," in *International Workshop on Intelligent Robots and Systems (IROS)*, 1991, pp. 1442–1447vol.3. doi: 10.1109/IROS.1991.174711.

[27] L. Schammer, S. Plambeck, F. H. Bahnsen, and G. Fey, "Learning models of cyber-physical systems using automata learning," in *Computers, Software, and Applications Conference (COMPSAC)*, 2021, pp. 1224–1229. doi: 10.1109/COMPSAC51774.2021.00169.

[28] S. Plambeck, J. Schyga, J. Hinckeldeyn, J. Kreutzfeldt, and G. Fey, "Automata learning for automated test generation of real time localization systems," in *Workshop on Learning in Control (LEAC) at CPS Week*, arxiv, 2021. doi: 10.48550/arXiv.2105.11911.

[29] A. Krumnow, S. Plambeck, and G. Fey, "Using forest structures for passive automata learning," in *Machine Learning for Cyber-physical Systems (ML4CPS)*, 2024, pp. 65–74. doi: 10.1007/978-3-031-47062-2_7.

[30] S. Plambeck, L. Schammer, and G. Fey, "On the viability of decision trees for learning models of systems," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022, pp. 696–701. doi: 10.1109/ASP-DAC52403.2022.9712579.

[31] S. Plambeck and G. Fey, "Decision tree models of continuous systems," in *International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–8. doi: 10.1109/ETFA52439.2022.9921491.

[32] S. Plambeck and G. Fey, "Regression trees for system models and prediction," in *Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis (OVERLAY) at AI*IA*, CEUR-WS.org, 2022, pp. 57–61.

[33] S. Plambeck and G. Fey, "Data-driven test generation for Black-box systems from learned decision tree models," in *International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023, pp. 27–32. doi: 10.1109/DDECS57882.2023.10139633.

[34] S. Plambeck, A. Bracht, N. Hranisavljevic, and G. Fey, "FaMoS — fast model learning for hybrid cyber-physical systems using decision trees," 2024. doi: 10.1145/3641513.3650131.

[35] S. Plambeck, M. Schmidt, A. Subias, L. Travé-Massuyès, and G. Fey, "Usability of symbolic regression for hybrid system identification - system classes and parameters," in *International Conference on Principles of Diagnosis and Resilient Systems (DX)*, in Open Access Series in Informatics (OASIcs). 2024, pp. 30:1–30:14. doi: 10.4230/OASIcs.DX.2024.30.

[36] S. Plambeck, M. Schmidt, G. Fey, A. Subias, and L. Travé-Massuyès, "Dynamics-based identification of hybrid systems using symbolic regression," in *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2024, pp. 64–71. doi: 10.1109/SEAA64295.2024.00019.

[37] M. Knitt, S. Plambeck, J. C. Wieck, J. Kohlisch, S. Balduin, E. M. Veith, J. Schyga, J. Hinckeldeyn, G. Fey, and J. Kreutzfeldt, "Towards the automatic generation of models for prediction, monitoring, and testing of cyber-physical systems," in *International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2023, pp. 1–4. doi: 10.1109/ETFA54631.2023.10275706.

[38] I. D. Mienye, Y. Sun, and Z. Wang, "Prediction performance of improved decision tree-based algorithms: A review," *Procedia Manufacturing*, vol. 35, pp. 698–703, 2019, doi: 10.1016/j.promfg.2019.06.011.

[39] G. Bakirtzis, E. Subrahmanian, and C. H. Fleming, "Compositional thinking in cyberphysical systems theory," *Computer*, vol. 54, no. 12, pp. 50–59, 2021, doi: 10.1109/MC.2021.3085532.

[40] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012, doi: 10.1109/JPROC.2011.2160929.

[41] E. VanDerHorn and S. Mahadevan, "Digital twin: Generalization, characterization and implementation," *Decision support systems*, vol. 145, p. 113524, 2021, doi: 10.1016/j.dss.2021.113524.

[42] M. Soori, B. Arezoo, and R. Dastres, "Digital twin for smart manufacturing, a review," *Sustainable Manufacturing and Service Economics*, vol. 2, p. 100017, 2023, doi: 10.1016/j.smse.2023.100017.

[43] M. Müller, N. Jazdi, A. Löcklin, L. Hettich, and M. Weyrich, "Adaptive models for safe maintenance planning of cyber-physical systems," in *CIRP Conference on Intelligent Computation in Manufacturing Engineering*, July 2021. doi: 10.1016/j.procir.2022.09.075.

[44] J. Jeon and G. Theotokatos, "A framework to assure the trustworthiness of physical model-based digital twins for marine engines," *Journal of Marine Science and Engineering*, vol. 12, no. 4, 2024, doi: 10.3390/jmse12040595.

[45] F. Vitale, S. Guarino, F. Flammini, L. Faramondi, N. Mazzocca, and R. Setola, "Process mining for digital twin development of industrial cyber-physical systems," *IEEE Transactions on Industrial Informatics*, 2024, doi: 10.1109/TII.2024.3465600.

[46] F. Aarts, F. Heidarian, H. Kuppens, P. Olsen, and F. Vaandrager, "Automata learning through counterexample guided abstraction refinement," 2012. doi: 10.1007/978-3-642-32759-9_4.

[47] V. Roussanaly, O. Sankur, and N. Markey, "Abstraction refinement algorithms for timed automata," in *Computer Aided Verification*, 2019, pp. 22–40. doi: 10.1007/978-3-030-25540-4_2.

[48] A. Devonport and M. Arcak, "Data-driven estimation of forward reachable sets," *Computation-aware Algorithmic Design for Cyber-physical Systems*. Springer International Publishing, Cham, pp. 165–185, 2023. doi: 10.1007/978-3-031-43448-8_8.

[49] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, Nov. 1984, doi: 10.1145/1968.1972.

[50] D. K. Chaturvedi, *Modeling and simulation of systems using MATLAB and simulink*. CRC press, 2017. doi: 10.1201/9781315218335.

[51] M. Tiller, *Introduction to physical modeling with modelica*, vol. 615. Springer Science & Business Media, 2012. doi: 10.1007/978-1-4615-1561-6.

[52] T. Weilkiens, "Chapter 4 - SysML—the systems modeling language," *Systems Engineering with SysML/UML*. in The MK/OMG Press. Morgan Kaufmann, Burlington, pp. 223–270, 2007. doi: 10.1016/B978-0-12-374274-2.00004-3.

[53] R. Barbau, C. Bock, and M. Dadfarnia, "Translator from extended SysML to physical interaction and signal flow simulation platforms, version 1.1," vol. 126, 2021, doi: 10.6028/jres.126.027.

[54] A. Ray and M. H. Kolekar, "Image segmentation and classification using deep learning," in *Machine Learning Algorithms for Signal and Image Processing*, John Wiley & Sons, Ltd, 2022, pp. 19–36. doi: 10.1002/9781119861850.ch2.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NIPS)*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017. doi: 10.5555/3295222.3295349.

[56] J. Rodellar, S. Alférez, A. Acevedo, A. Molina, and A. Merino, "Image processing and machine learning in the morphological analysis of blood cells," *International Journal of Laboratory Hematology*, vol. 40, no. S1, pp. 46–53, 2018, doi: 10.1111/ijlh.12818.

[57] Z.-H. Zhou, "Ensemble learning," in *Machine Learning*, Singapore: Springer Singapore, 2021, pp. 181–210. doi: 10.1007/978-981-15-1967-3_8.

[58] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, "Autogluon-tabular: Robust and accurate Automl for structured data." arXiv, 2020. doi: 10.48550/arXiv.2003.06505.

[59] A. Graß, C. Beecks, and J. A. C. Soto, "Unsupervised anomaly detection in production lines," in *Machine Learning for Cyber Physical Systems (ML4CPS)*, 2018. doi: 10.1007/978-3-662-58485-9_3.

[60] F. H. Bahnsen, J. Kaiser, and G. Fey, "Designing recurrent neural networks for monitoring embedded devices," in *IEEE European Test Symposium (ETS)*, 2021. doi: 10.1109/ETS50041.2021.9465460.

[61] H. Steude, A. Windmann, and O. Niggemann, "Learning physical concepts in CPS: A case study with a Three-tank system," *IFAC-PapersOnLine*, vol. 55, no. 6, pp. 15–22, 2022, doi: https://doi.org/10.1016/j.ifacol.2022.07.099.

[62] E. M. Veith, S. Balduin, N. Wenninghoff, T. Wolgast, M. Baumann, D. Winkler, L. Hammer, A. Salman, M. Schulz, A. Raeiszadeh, T. Logemann, and A. Wellßow, "Palaestrai: A training ground for autonomous agents," in *European Simulation and Modelling Conference (EMS)*, 2023, pp. 199–204.

[63] Z. Wang, Z. Zhou, W. Xu, C. Sun, and R. Yan, "Physics informed neural networks for fault severity identification of axial piston pumps," *Journal of Manufacturing Systems*, vol. 71, pp. 421–437, Dec. 2023, doi: 10.1016/j.jmsy.2023.10.002.

[64] E. Kiyani, K. Shukla, G. E. Karniadakis, and M. Karttunen, "A framework based on symbolic regression coupled with Extended Physics-informed neural networks for gray-box learning of equations of motion from data," *Computer Methods in Applied Mechanics and Engineering*, vol. 415, p. 116258, 2023, doi: 10.1016/j.cma.2023.116258.

[65] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021, doi: 10.1038/s42254-021-00314-5.

[66] E. M. Gold, "Language identification in the limit," *Information and Control*, vol. 10, no. 5, pp. 447–474, 1967, doi: 10.1016/S0019-9958(67)91165-5.

[67] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987, doi: 10.1016/0890-5401(87)90052-6.

[68] M. Merten, "Active automata learning for real life applications," Doctoral dissertation, 2013. doi: 10.17877/DE290R-5169.

[69] H. Urbat and L. Schröder, "Automata learning: An algebraic approach," in *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, Saarbrücken, Germany, 2020, pp. 900–914. doi: 10.1145/3373718.3394775.

[70] K. P. Murphy, "Passively learning finite automata," technical report, 1995.

[71] A. Maier, "Online passive learning of timed automata for cyber-physical production systems," in *IEEE International Conference on Industrial Informatics (INDIN)*, 2014, pp. 60–66. doi: 10.1109/INDIN.2014.6945484.

[72] S. Linderman, M. Johnson, A. Miller, R. Adams, D. Blei, and L. Paninski, "Bayesian learning and inference in recurrent switching linear dynamical systems," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2017.

[73] O. Niggemann, B. Stein, A. Vodencarevic, A. Maier, and H. K. Büning, "Learning behavior models for hybrid timed systems," in *AAAI Conference on Artificial Intelligence*, 2012, pp. 250–264. doi: 10.1609/aaai.v26i1.8296.

[74] O. Maler and G. Batt, "Approximating continuous systems by timed automata," 2008, pp. 77–89. doi: 10.1007/978-3-540-68413-8_6.

[75] M. Tappler, B. K. Aichernig, K. G. Larsen, and F. Lorber, "Time to learn – learning timed automata from tests," in *Formal Modeling and Analysis of Timed Systems*, 2019. doi: 10.1007/978-3-030-29662-9_13.

[76] M. J. Kearns, *An introduction to computational learning theory*. Cambridge, Mass. u.a.: MIT Press, 1994. doi: 10.7551/mitpress/3897.001.0001.

[77] D. Angluin and P. D. Laird, "Learning from noisy examples," *Machine Learning*, vol. 2, pp. 343–370, 1988, doi: 10.1007/BF00116829.

[78] R. Carrasco and J. Oncina, "Learning deterministic regular grammars from stochastic samples in polynomial time," *RAIRO - Theoretical Informatics and Applications*, vol. 33, 1999, doi: 10.1051/ita:1999102.

[79] C. de la Higuera, *Grammatical inference: Learning automata and grammars*. Cambridge University Press, 2010. doi: 10.1017/CBO9781139194655.

[80] D.-Y. Yeung and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," *Pattern Recognition*, vol. 36, no. 1, pp. 229–243, 2003, doi: 10.1016/S0031-3203(02)00026-2.

[81] H. Mao, Y. Chen, M. Jaeger, T. Nielsen, K. Larsen, and B. Nielsen, "Learning probabilistic automata for model checking," 2011, pp. 111–120. doi: 10.1109/QEST.2011.21.

[82] M. Schmidt, S. Plambeck, and G. Fey, "Learnability of models for cyber-physical systems – a review," *Design and Verification of Cyber-physical Systems: From Theory to Applications*. 2025.

[83] J. Jeon and G. Theotokatos, "A framework to assure the trustworthiness of physical model-based digital twins for marine engines," *Journal of Marine Science and Engineering*, vol. 12, no. 4, 2024, doi: 10.3390/jmse12040595.

[84] X. Xin, S. L. Keoh, M. Sevegnani, M. Saerbeck, and T. P. Khoo, "Adaptive model verification for modularized Industry 4.0 applications," *IEEE Access*, vol. 10, pp. 125353–125364, 2022, doi: 10.1109/ACCESS.2022.3225399.

[85] Z. Wang, Z. Zhou, W. Xu, C. Sun, and R. Yan, "Physics informed neural networks for fault severity identification of axial piston pumps," *Journal of Manufacturing Systems*, vol. 71, pp. 421–437, Dec. 2023, doi: 10.1016/j.jmsy.2023.10.002.

[86] H. Wang, Y. Qamsane, J. Moyne, and K. Barton, "Merging subject matter expertise and deep convolutional neural network for state-based online machine-part interaction classification," in *Manufacturing Science & Engineering Conference (MSEC)*, June 2021. doi: 10.1115/MSEC2021-63661.

[87] G. Dang and D. Wang, "Recurrent stochastic configuration networks with incremental blocks," no. arXiv:2411.11303. arXiv, Nov. 2024. doi: 10.48550/arXiv.2411.11303.

[88] Y. Zeng, P. Thiyagarajan, B. Chan, and R. Jin, "Synthetic data generation and sampling for online training of DNNs in manufacturing supervised learning problems," in *IEEE International Conference on Automation Science and Engineering*, 2023. doi: 10.1109/CASE56687.2023.10260330.

[89] F. Su, S. Cao, T. Hai, and J. Yuan, "Multiscale redundant second generation wavelet kernel-driven convolutional neural network for rolling bearing fault diagnosis," in *International Conference on Neural Computing for Advanced Applications*, H. Zhang, Y. Ke, Z. Wu, T. Hao, Z. Zhang, W. Meng, and Y. Mu, Eds., Singapore: Springer Nature, 2023, pp. 263–278. doi: 10.1007/978-981-99-5847-4_19.

[90] G. Saveri and L. Bortolussi, "Retrieval-augmented mining of temporal logic specifications from data," in *Machine Learning and Knowledge Discovery in Databases*, A. Bifet, J. Davis, T. Krilavičius, M. Kull, E. Ntoutsi, and I. Zliobaitė, Eds., Cham: Springer Nature Switzerland, 2024, pp. 315–331. doi: 10.1007/978-3-031-70368-3_19.

[91] G. Chen and Z. Kong, "Data-driven approximate abstraction for Black-box piecewise affine systems," in *Annual American Control Conference (ACC)*, June 2018, pp. 786–791. doi: 10.23919/ACC.2018.8431327.

[92] A. Brusaferri, M. Matteucci, and S. Spinelli, "Identification of probability weighted ARX models with arbitrary domains," no. arXiv:2009.13975. arXiv, Sept. 2020. doi: 10.48550/arXiv.2009.13975.

[93] Silvia Maria Zanoli, Luca Barboni, and Tommaso Leo, "An application of hybrid automata for the MIMO model identification of a gasification plant," in *IEEE International Conference on Systems, Man and Cybernetics*, Montreal, QC, Canada: IEEE, 2007, pp. 1427–1432. doi: 10.1109/ICSMC.2007.4413764.

[94] M. Waga, E. Castellano, S. Pruekprasert, S. Klikovits, T. Takisaka, and I. Hasuo, "Dynamic shielding For~reinforcement learning In~black-box environments," in *Automated Technology for Verification and Analysis*, A. Bouajjani, L. Holík, and Z. Wu, Eds., Cham: Springer International Publishing, 2022, pp. 25–41. doi: 10.1007/978-3-031-19992-9_2.

[95] A. Devonport and M. Arcak, "Data-driven estimation of forward reachable sets," *Computation-aware Algorithmic Design for Cyber-physical Systems*. Springer International Publishing, Cham, pp. 165–185, 2023. doi: 10.1007/978-3-031-43448-8_8.

[96] C. Sun, Y. Cui, Y. Lu, Y. Cao, D. Cao, and A. Khajepour, "Robust learning for autonomous driving perception tasks in cyber-physical-social systems," in *IEEE International Conference on Digital Twins and Parallel Intelligence DTPI*, 2023. doi: 10.1109/DTPI59677.2023.10365413.

[97] F. Vitale, S. Guarino, F. Flammini, L. Faramondi, N. Mazzocca, and R. Setola, "Process mining for digital twin development of industrial cyber-physical systems," *IEEE Transactions on Industrial Informatics*, 2024, doi: 10.1109/TII.2024.3465600.

[98] G. Prasad, G. Leng, T. McGinnity, and D. Coyle, "Online identification of self-organizing fuzzy neural networks for modeling time-varying complex systems," *Evolving Intelligent Systems*. Wiley, pp. 201–228, Mar. 2010. doi: 10.1002/9780470569962.ch9.

[99] E. Barrena Algara, "Soft operators decision trees: Uncertainty and stability related issues," Doctoral dissertation, 2008.

[100] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014. doi: 10.1017/CBO9781107298019.

[101] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers - a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 4, pp. 476–487, 2005, doi: 10.1109/TSMCC.2004.843247.

[102] G. Gopalakrishnan, "Machines, languages, dfa," in *Computation engineering: Applied automata theory and logic*, Springer Science & Business Media, 2006. doi: 10.1007/0-387-32520-4_8.

[103] J. Oncina and P. Garcia, "Identifying regular languages in polynomial time," *Advances in Structural and Syntactic Pattern Recognition*, vol. 5, pp. 99–108, 1993, doi: 10.1142/9789812797919_0007.

[104] C. De La Higuera, "Characteristic sets for polynomial grammatical inference," *Machine Learning*, vol. 27, pp. 125–138, 1997, doi: 10.1023/A:1007353007695.

[105] P. García, D. López, and M. V. de Parga, "Polynomial characteristic sets for DFA identification," *Theoretical Computer Science*, vol. 448, pp. 41–46, 2012, doi: 10.1016/j.tcs.2012.04.042.

[106] A. Bainczyk, A. Schieweck, B. Steffen, and F. Howar, "Model-based testing without models: The TodoMVC case study," in *Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday*, J.-P. Katoen, R. Langerak, and A. Rensink, Eds., Cham: Springer International Publishing, 2017, pp. 125–144. doi: 10.1007/978-3-319-68270-9_7.

[107] van Beek, Jansen, Rooda, Schiffelers, Man, and Reniers, "Relating Chi to hybrid automata," in *Winter Simulation Conference*, 2003, pp. 632–640. doi: 10.1109/WSC.2003.1261478.

[108] M. Müller, "Dynamic time warping," in *Information retrieval for music and motion*, Heidelberg: Springer-Verlag, 2007, pp. 69–84. doi: 10.1007/978-3-540-74048-3_4.

[109] I. Saberi, F. Faghih, and F. S. Bavil, "A passive online technique for learning hybrid automata from Input/output traces," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 1, pp. 1–24, 2021, doi: 10.1145/3556543.

[110] J. R. Koza, *On the programming of computers by means of natural selection.* in Koza, John R. Genetic programming. Cambridge, Mass. u.a.: MIT Press, 1992.

[111] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*, vol. 10. Springer, 2008. doi: 10.1007/s10710-008-9073-y.

[112] G. Kronberger, L. Kammerer, and M. Kommenda, "Identification of dynamical systems using symbolic regression," in *Computer Aided Systems Theory (EUROCAST)*, R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, Eds., 2020, pp. 370–377. doi: 10.1007/978-3-030-45093-9_45.

[113] D. Kantor, F. J. Von Zuben, and F. O. de Franca, "Simulated annealing for symbolic regression," in *Proceedings of the genetic and evolutionary computation conference (GECCO)*, 2021, pp. 592–599. doi: 10.1145/3449639.345934.

[114] B. Steffen, F. Howar, and M. Merten, "Introduction to active automata learning from a practical perspective," in *Formal Methods for Eternal Networked Software Systems: International School on Formal Methods for the Design of Computer (SFM)*, M. Bernardo and V. Issarny, Eds., Berlin, Heidelberg: Springer, 2011. doi: 10.1007/978-3-642-21455-4_8.

[115] M. Isberner, F. Howar, and B. Steffen, "The Open-source Learnlib - A framework for active automata learning," in *Computer Aided Verification (CAV)*, D. Kroening and C. S. Pasareanu, Eds., Springer, 2015, pp. 487–495. doi: 10.1007/978-3-319-21690-4_32.

[116] B. O. Bouamama, R. M. Alaoui, P. Taillibert, and M. Staroswiecki, "Diagnosis of a two-tank system," technical report, 2001.

[117] MathWorks, "Bang-bang control using temporal logic." Accessed: Apr. 19, 2025. [Online]. Available: https://de.mathworks.com/help/stateflow/ug/bang-bang-control-using-temporal-logic.html

[118] N. Zaupa, L. Martínez-Salamero, C. Olalla, and L. Zaccarian, "Hybrid control of Self-oscillating resonant converters," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 2, pp. 881–888, 2023, doi: 10.1109/tcst.2022.3179948.

[119] O. A. Beg, H. Abbas, T. T. Johnson, and A. Davoudi, "Model validation of PWM DC-DC converters," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 9, pp. 7049–7059, 2017, doi: 10.1109/TIE.2017.2688961.

[120] X. Yang, O. A. Beg, M. Kenigsberg, and T. T. Johnson, "A framework for identification and validation of affine hybrid automata from Input-output traces," *ACM Transactions on Cyber-Physical Systems*, vol. 6, no. 2, pp. 1–24, 2022, doi: 10.1145/3470455.

[121] F. H. Bahnsen and G. Fey, "Local monitoring of embedded applications and devices using artificial neural networks," in *Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 485–491. doi: 10.1109/DSD.2019.00076.

[122] P. Dupont, L. Miclet, and E. Vidal, "What is the search space of the regular inference?," in *Grammatical Inference and Applications (ICGI)*, 1994, pp. 25–37. doi: 10.1007/3-540-58473-0_134.

[123] J. Oncina and P. García, "Inferring regular languages in polynomial updated time," in *Pattern Recognition and Image Analysis*, pp. 49–61. doi: 10.1142/9789812797902_0004.

[124] M. Isberner, F. Howar, and B. Steffen, "The TTT algorithm: A Redundancy-free approach to active automata learning," in *Runtime Verification (RV)*, 2014, pp. 307–322. doi: 10.1007/978-3-319-11164-3_26.

[125] K. J. Lang, B. A. Pearlmutter, and R. A. Price, "Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm," in *Grammatical Inference (ICGI)*, V. Honavar and G. Slutzki, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 1–12. doi: 10.1007/BFb0054059.

[126] J. Abela, F. Coste, and S. Spina, "Mutually compatible and incompatible merges for the search of the smallest consistent dfa," in *Grammatical Inference: Algorithms and Applications (ICGI)*, 2004, pp. 28–39. doi: 10.1007/978-3-540-30195-0_4.

[127] K. Meinke, "Learning-based testing of cyber-physical Systems-of-systems: A platooning study," in *Computer Performance Engineering (EPEW)*, P. Reinecke and A. Di Marco, Eds., 2017, pp. 135–151. doi: 10.1007/978-3-319-66583-2_9.

[128] A. Rasool, G. Alpár, and J. de Ruiter, "State machine inference of QUIC." 2019. doi: https://doi.org/10.48550/arXiv.1903.04384.

[129] M. Tappler, S. Pranger, B. Könighofer, E. Muškardin, R. Bloem, and K. Larsen, "Automata learning meets shielding," in *Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles (ISoLA)*, T. Margaria and B. Steffen, Eds., Springer International Publishing, 2022, pp. 335–359. doi: 10.1007/978-3-031-19849-6_20.

[130] G. Weiss, Y. Goldberg, and E. Yahav, "Extracting automata from recurrent neural networks using queries and counterexamples (extended version)," *Machine Learning*, vol. 113, no. 5, pp. 2877–2919, 2024, doi: 10.1007/s10994-022-06163-2.

[131] T. S. Chow, "Testing software design modeled by Finite-state machines," *IEEE Transactions on Software Engineering*, no. 3, pp. 178–187, 1978, doi: 10.1109/TSE.1978.231496.

[132] G. Cicala, A. Khalili, G. Metta, L. Natale, S. Pathak, L. Pulina, and A. Tacchella, "Engineering approaches and methods to verify software in autonomous systems," *Advances in Intelligent Systems and Computing*, vol. 302, pp. 1683–1700, 2016, doi: 10.1007/978-3-319-08338-4_121.

[133] P. García, M. Vázquez de Parga, D. López, and J. Ruiz, "Learning automata teams," in *Grammatical Inference: Theoretical Results and Applications (ICGI)*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 52–65. doi: 10.1007/978-3-642-15488-1_6.

[134] S. Shoham, E. Yahav, S. J. Fink, and M. Pistoia, "Static specification mining using automata-based abstractions," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 651–666, 2008, doi: 10.1109/TSE.2008.63.

[135] S. Cassel, F. Howar, B. Jonsson, and B. Steffen, "Active learning for extended finite state machines," *Formal Aspects of Computing*, vol. 28, no. 2, pp. 233–263, 2016, doi: 10.1007/s00165-016-0355-5.

[136] M. Utting and B. Legeard, "Chapter 5 - testing from finite state machines," *Practical Model-Based Testing*. Morgan Kaufmann, San Francisco, pp. 139–185, 2007. doi: 10.1016/B978-012372501-1/50006-5.

[137] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the Robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016, doi: 10.1109/TRO.2016.2624754.

[138] C. Hansen, D. Gibas, J.-L. Honeine, N. Rezzoug, P. Gorce, and B. Isableu, "An inexpensive solution for motion analysis," *Proceedings of the Institution of Mechanical Engineers, Part P: Journal of Sports Engineering and Technology*, vol. 228, 2014, doi: 10.1177/1754337114526868.

[139] ISO/IEC 18305:2016, "Information technology - real time locating systems - test and evaluation of localization and tracking systems," Standard, Nov. 2016.

[140] L. Welch, "The generalization of `student's` problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1–2, pp. 28–35, 1947, doi: 10.1093/biomet/34.1-2.28.

[141] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N.

Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.

[142] E. Lucena-Sanchez, G. Sciavicco, and I. E. Stan, "Symbolic learning with interval temporal logic: Thecase of regression," in *Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY)*, Bozen Bolzano: CEUR, 2020, pp. 5–10.

[143] L. Goupil, L. Travé-Massuyès, E. Chanthery, T. Kohler, and S. Delautier, "Tree based diagnosis enhanced with meta knowledge applied to dynamic systems," *IFAC-PapersOnLine*, vol. 58, no. 4, pp. 1–6, 2024, doi: 10.1016/j.ifacol.2024.07.184.

[144] G.-V. Jourdan, H. Ural, and H. Yenigün, "Reduced checking sequences using unreliable reset," *Information Processing Letters*, vol. 115, pp. 532–535, 2015, doi: 10.1016/j.ipl.2015.01.002.

[145] R. Groz, A. Simao, A. Petrenko, and C. Oriat, "Inferring finite state machines without reset using state identification sequences," in *Testing Software and Systems (ICTSS)*, 2015, pp. 161–177. doi: 10.1007/978-3-319-25945-1_10.

[146] J. Mordeson and D. Malik, *Fuzzy automata and languages: Theory and applications*. Chapman, Hall/CRC, 2002, pp. 1–556. doi: 10.1201/9781420035643.

[147] M. Češka, V. Havlena, L. Holík, O. Lengál, and T. Vojnar, "Approximate reduction of finite automata for High-speed network intrusion detection," *International Journal on Software Tools for Technology Transfer*, pp. 523–539, 2019, doi: 10.1007/s10009-019-00520-8.

[148] MathWorks, "Fitcnet - train neural network classification model." Accessed: June 06, 2025. [Online]. Available: https://de.mathworks.com/help/stats/fitcnet.html

[149] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 2006. doi: 10.1007/978-0-387-40065-5.

[150] MathWorks, "Fitrnet - train neural network regression model." Accessed: Sept. 02, 2025. [Online]. Available: https://de.mathworks.com/help/stats/fitrnet.html

[151] S. Biswas and R. D. Blanton, "Statistical test compaction using binary decision trees," *IEEE Design & Test of Computers*, vol. 23, no. 6, pp. 452–462, 2006, doi: 10.1109/MDT.2006.154.

[152] M. R. Garey, *Computers and intractability : a guide to the theory of Np-completeness*. in A series of books in the mathematical sciences. New York, NY: W. H. Freeman, Company, 1979.

[153] B. Korte, *Combinatorial optimization: Theory and algorithms*. Berlin, Heidelberg: Springer, 2006. doi: 10.1007/978-3-540-71844-4_16.

[154] N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales-Palacio, "DyClee: Dynamic clustering for tracking evolving environments," *Pattern Recognition*, vol. 94, pp. 162–186, 2019, doi: 10.1016/j.patcog.2019.05.024.

[155] S. Gaucel, M. Keijzer, E. Lutton, and A. Tonda, "Learning dynamical systems using standard symbolic regression," in *Genetic Programming (EuroGP)*, 2014, pp. 25–36. doi: 10.1007/978-3-662-44303-3_3.

[156] D. L. Ly and H. Lipson, "Learning symbolic representations of hybrid dynamical systems," *Journal of Machine Learning Research*, vol. 13, no. 115, pp. 3585–3618, 2012.

[157] N. Kochdumper, M. A. Foughali, P. Habermehl, and E. Asarin, "Robust identification of hybrid automata from noisy data," in *ACM International Conference on Hybrid Systems: Computation and Control*, 2025. doi: 10.1145/3716863.3718030.

[158] P. Cull, M. Flahive, and R. Robson, "Matrix difference equations," in *Difference Equations: From Rabbits to Chaos*, New York: Springer, 2005. doi: 10.1007/0-387-27645-9_7.

[159] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, no. 4, 2020, doi: 10.1016/j.sigpro.2019.107299.

[160] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, June 1981, doi: 10.1145/358669.358692.

[161] M. Cranmer, "Interpretable machine learning for science with Pysr and symbolicregression.jl." 2023.

[162] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework." 2019.

[163] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *International Conference on Machine Learning (ICML)*, 2014. doi: 10.5555/3044805.3044891.

[164] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[165] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, High-performance deep learning library," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. doi: 10.5555/3454287.3455008.

[166] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and others, "Tensorflow: A system for large-scale machine learning," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265–283. doi: 10.5555/3026877.3026899.

[167] O. Niggemann, G. Biswas, J. S. Kinnebrew, H. Khorasgani, S. Volgmann, and A. Bunte, "Data-driven monitoring of cyber-physical systems leveraging on big data and the Internet-of-things for diagnosis and control," in *International Workshop on Principles of Diagnosis (DX)*, 2015, pp. 185–192.

[168] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, 2000, doi: 10.1109/5.871304.

[169] E. A. Lee, "Fundamental limits of cyber-physical systems modeling," *ACM Transactions on Cyber-Physical Systems*, vol. 1, no. 1, Nov. 2016, doi: 10.1145/2912149.

[170] M. Bayoudh and L. Travé-Massuyès, "Diagnosability analysis of hybrid systems cast in a discrete-event framework," *Discrete Event Dynamic Systems*, vol. 24, no. 3, pp. 309–338, 2014, doi: 10.1007/s10626-012-0153-z.

[171] M. Schmidt, S. Plambeck, and G. Fey, "Learnability of models for cyber-physical systems – a review," *Design and Verification of Cyber-Physical Systems: From Theory to Applications*. 2025.

[172] S. Plambeck, N. Hranisavljevic, M. Schmidt, K. Balzereit, A. Bracht, M. Redeker, A. Arabizadeh Negar Diedrich, J. Eickmeier, O. Niggemann, and G. Fey, "Workshop report: Learning approaches for hybrid dynamical systems," in *Machine Learning For Cyber-Physical Systems (ML4CPS)*, 2025. doi: 10.24405/20029.

Index of Figures

Figure 1 Illustration of the motivating example: a LiDAR localization system on a forklift in a logistics warehouse.[3] 3
Figure 2 Modeling process for the RTLS: transforming continuous measurements into categorical predictions. DTL finds a model, which describes the dependency between the external influences and the localization accuracy. 5
Figure 3 Modeling process for the RTLS with RTL: directly predicting continuous localization errors without categorical abstraction. 6
Figure 4 Decision tree model for application-related categories (storage box-precise, medium object-precise, and small object-precise). 7
Figure 5 Decision tree model with RTL. The predicted localization error is shown in the leaves of the tree. 8
Figure 6 Data-driven model learning: the observations of the system are used to learn a model. A learning algorithm is applied to the observations. The learning algorithm and the model structure have a mutual dependency — either a learning algorithm for a specific model structure or a generic learning algorithm that applies to different model structures is used. 10
Figure 7 Three types of model structures (green) FSMs, decision trees, and hybrid automata are considered in this work. The respective learning algorithms (pink) for these model structures are shown per column. A joined comparison and discussion accumulates the results. 10
Figure 8 Spectrum of data-driven influence in modeling of CPSs [82] with respect to domain and expert knowledge. The figure illustrates the varying degrees of reliance on data-driven methods. On the left end, models are derived almost entirely through domain expertise and manual design. The right end represents models that are generated predominantly from

[3] Image generated with ChatGPT

data-driven approaches. Hybrid and physics-informed methods occupy the middle ground, leveraging both data and domain knowledge. 23
Figure 9 Example of a decision tree. The tree is used to classify observations based on the values of the features. The inner nodes and edges are associated with features and split conditions, while the leaves are associated with class labels. 28
Figure 10 Example of a DFA with five states and the input alphabet $\Sigma = \{0, 1\}$. State 1 is the initial state and State 4 is an accepting state of the automaton. 30
Figure 11 Mealy machine representation of a coffee machine. The input alphabet is $I = \{\texttt{BUTTON}, \texttt{CLEAN}, \texttt{POD}, \texttt{WATER}\}$ and the output alphabet is $O = \{\texttt{ok}, \texttt{error}, \texttt{coffee}\}$. 31
Figure 12 Active automata learning process. The learner refines the model using membership queries and equivalence queries [106]. 33
Figure 13 Hybrid automaton representation of a boiler system. The system has two modes: the `On` mode (heating) and the `Off` mode (cooling), which are described by differential equations [107]. 35
Figure 14 DTW comparison of two sines. DTW aligns the two signals despite their different sampling rates and lengths. 37
Figure 15 Genetic programming algorithm for symbolic regression. The algorithm starts with a population of random expressions and iteratively refines them using genetic operators such as selection, mutation, and recombination. 39
Figure 16 Illustration of the examples used in this thesis.[4] 40
Figure 17 Sub-components of the coffee machine. The sub-components are the water tank with an optional extension to a two-tank system, the boiler, and the power converter. 40
Figure 18 Water tank system with inflow pump i_1, outflow pump i_2 for flushing operation and outflow valve for draining operation. The cross-sectional area of the tank is A and the height of the water in the tank is h. 42
Figure 19 Signals of the flushing operation of the water tank. The input signals are the states (1 or 0) of the valves at inflow i_1 and outflow i_2 pumps, while the output o is the water level of the reservoir in dm. 42
Figure 20 Mealy machine representation of the water tank system with draining operation. The system has two input symbols `fill` and `drain` and the output symbols `1,2,3,4`, representing the water level of the tank. 43
Figure 21 Two-tank system for water storage at the water supply and treatment center (Versorgungsbetriebe Helgoland) at Helgoland, Germany. 43

[4] Image generated with ChatGPT

Figure 22 Boiler system with controlled temperature range. The system has two modes: the heating mode and the cooling mode, which are described by differential equations. 44
Figure 23 Power converter consisting of a capacitor C, an inductance L, and a resistor R with parallel (top) and serial (bottom) configuration. The system has a controlled input voltage $v_s = \sigma V_g$ where V_g is a constant input and σ is a switching variable, which is either 1 or -1 [118]. 46
Figure 24 Electrical circuit of a buck converter [119]. 46
Figure 25 Hybrid automaton representation of the buck converter [119]. The modes are governed by linear differential equations. The events for switching between the modes depends on the duty cycle D, the switching period T and the inductor current i_L. At every transition, the time t is reset to zero. 46
Figure 26 Example trace of the vacuum cleaner robot. The robot moves inside a room (gray area) surrounded by walls (black lines). The red line shows the trajectory of the robot. 47
Figure 27 Automaton model of the vacuum cleaner robot in random autonomous cleaning mode. The transitions between the states are labeled with the actions taken by the robot. The actions form the alphabet $\Sigma = \{\texttt{move}, \texttt{touch}, \texttt{rotate}, \texttt{backwards}\}$. State 0, State 2, State 3, and State 4 are accepting states representing the normal operation of the robot, while State 1 is non-accepting and represents all deviations from the normal operation. 47
Figure 28 Overview of the presented models (green) and learning approaches (pink) per model type in this work. Chapter 4 covers passive and active automata learning as well as automata forests. 49
Figure 29 An abstract representation of an SUL. The concrete inputs and outputs (hatched) are abstracted to abstract inputs and outputs (solid). This abstraction relates the concrete system and the abstract model. 52
Figure 30 Example of sub-spaces for the two dimensions x and y with categorization spaces of width and height 5. 53
Figure 31 Example for the discretization of a continuous output trace o with a sampling period t_S. The continuous values are discretized to four output symbols $o_1 = \texttt{1}$, $o_2 = \texttt{2}$, $o_3 = \texttt{3}$, and $o_4 = \texttt{4}$, indicated in orange above the trace. 54
Figure 32 Learned automaton model of the vacuum cleaner robot in random autonomous cleaning mode. The input alphabet is $\Sigma = \{\texttt{move}, \texttt{touch}, \texttt{rotate}, \texttt{backwards}\}$ and State 0 is the accepting state. 57
Figure 33 Learned automaton model of the vacuum cleaner robot in controlled cleaning mode and input alphabet

$\Sigma = \{\texttt{move}, \texttt{touch}, \texttt{rotate}, \texttt{backwards}\}$ and States 0, 2, 3, and 4 are accepting states. 58
Figure 34 Learned automaton model of the vacuum cleaner robot in controlled cleaning mode. The input alphabet is $\Sigma = \{\texttt{move}, \texttt{rotate}\}$ and States 0, 1, 2 are accepting states. The move command has a distance of 1 m and the rotate command has a rotation angle of 180°. 59
Figure 35 Learned automaton model of the vacuum cleaner robot in controlled cleaning mode. The input alphabet is $\Sigma = \{\texttt{move}, \texttt{rotate}\}$ and States 0, 1, 2, 4, 5 are accepting states. The move command has a distance of 0.5 m and the rotate command has a rotation angle of 180°. 59
Figure 36 Learned automaton model of the vacuum cleaner robot in controlled cleaning mode. The input alphabet is $\Sigma = \{\texttt{move}, \texttt{rotate}\}$ and all states except for State 6 are accepting states. The move command has a distance of 0.25 m and the rotate command has a rotation angle of 180°. 59
Figure 37 Layout of the experimental setup for the case study at the Institute for Logistics Engineering at Hamburg University of Technology. The reference localization system is indicated by the cameras in the corners of the testbed. The RTLS is installed on a trolley, which is manually pushed along a predefined trajectory. 62
Figure 38 Learned Mealy machine for the localization system. The automaton has three states $Q = \{\texttt{0}, \texttt{1}, \texttt{2}\}$, four input symbols $I = \{\texttt{N}, \texttt{E}, \texttt{S}, \texttt{W}\}$ and two output symbols $O = \{\texttt{a}, \texttt{b}\}$. 65
Figure 39 Concept of automata forests: the black dot represents the target automaton, the blue dots represent the instances in the automata forest and the red dot represents the automaton that is determined with a traditional automata learning algorithm. 70
Figure 40 Histogram over 1000 repetitions of the number of correctly translated traces for the coffee machine example on 1000 trials. 72
Figure 41 Histogram over 1000 repetitions of the number of correctly translated traces for the artificial Mealy machine with 10 states on 1000 trials. 72
Figure 42 Histogram over 1000 repetitions of the number of correctly translated traces for the artificial Mealy machine with 15 states on 1000 trials. 72
Figure 43 Histogram over 1000 repetitions of the number of correctly translated traces for the artificial Mealy machine with 20 states on 1000 trials. 72
Figure 44 Histogram over 1000 repetitions of the number of correctly translated traces for the artificial Mealy machine with 25 states on 1000 trials. 72
Figure 45 Overview of the presented models (green) and learning approaches (pink) per model type in this work. This chapter presents decision tree models for CPS. 77
Figure 46 Illustration of the decision functions for language-based and output-based decision tree models. 81

Figure 47 Mealy machine with alphabet $\Sigma = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}\}$. The mapping to input-output pairs is given in the table on the left. The system is not learnable from bounded observations because of the two self-loops on the same symbol $\mathcal{B}$. 84
Figure 48 Architecture of the neural network used for comparison with the decision tree model. The input layer has n neurons, where n is the length of the bounded history. The output layer has one neuron for each possible output symbol of the system. 89
Figure 49 Neural network architecture for regression tasks. The network consists of an input layer, two hidden layers with ReLU activations, and an output layer. 95
Figure 50 Decision tree model of the FSM from Figure 47 with $n = 2$. 98
Figure 51 FSM representation of the decision tree model from Figure 50. 98
Figure 52 Decision tree model of the coffee machine from Figure 11 with $n = 2$. .104
Figure 53 Overview of the presented models (green) and learning approaches (pink) per model type in this work. This chapter presents approaches for learning hybrid automata models. 109
Figure 54 Learning process for hybrid automata models consisting of four steps. 111
Figure 55 Hybrid decision tree consisting of a decision tree (left) and flow functions (right). The modes are indicated by colors. 113
Figure 56 Illustration of the mapping w for the segmentation objective Ω_{seg}. The ground truth transition points are indicated in red, while the found transition points are blue. The mapping for this example is $w = \{\langle 1, 1\rangle, \langle 2, 2\rangle, \langle 3, \emptyset\rangle, \langle 4, 3\rangle, \langle 5, 5\rangle, \langle 6, 6\rangle, \langle 7, 8\rangle\}$. 114
Figure 57 Example of the trace segmentation step in FaMoS using $\omega = 5$ and a sampling rate of $1/100$. The scaled deviation function is shown in blue. The detected transition points are shown as vertical dashed lines. 117
Figure 58 Trace segmentation on the gradient of o_1 representing an example of a numerically calculated higher order derivative. 118
Figure 59 Adaptation of DTW for FaMoS: the segments are either aligned at the start or at the end. The minimum of the two similarities is used as the final similarity. 118
Figure 60 Grouped segments for three traces $\boldsymbol{o_1}$, $\boldsymbol{o_2}$, and $\boldsymbol{o_3}$. Every trace is grouped individually. The color indicates the group of the segment. 121
Figure 61 Example of global grouping of segments. Global groups are formed by combining the group IDs of all variables. The example shows three signals with three groups each. The global groups are shown in the last row. The colors indicate the group IDs of the segments. 121
Figure 62 Predicted traces of the two tank system for HAutLearn (dotted-red) and FaMoS (dashed-green) in comparison to the ground truth trace (blue). . 127

Figure 63 Predicted traces for the converter for HAutLearn (dotted-red) and FaMoS (dashed-green) in comparison to the ground truth trace (blue). 128
Figure 64 Predicted traces for the boiler for HAutLearn (dotted-red) and FaMoS (dashed-green) in comparison to the ground truth trace (blue). 128
Figure 65 Excerpt of trace segmentation with SR on an example trace. 134
Figure 66 Parameter importances of the two tank system with substitution (green), two tank system without substitution (red), and power converter (blue). 139
Figure 67 Segmentation and grouping for the converter example: colors indicate the group of the segment. 140
Figure 68 Predicted (red) and ground truth (blue) evaluation traces. 141
Figure 69 Overview of the presented models (green) and learning approaches (pink) per model type in this work. The focus of this chapter is the comparison and assessment of the methods. 145
Figure 70 General learning process for CPS-driven applications consisting of seven steps. 150
Figure 71 Component-view of the concepts, showing the three steps of data-driven modeling (A to C) extended by an evaluation (D to F). 152
Figure 72 Flowchart for the learning strategies offline, incremental, and active learning. 153
Figure 73 Illustration of modularization in Flowcean composing the learning and evaluation pipeline. 153

Index of Tables

Table I Overview of the research questions and their corresponding chapters and publications. 11
Table II Concepts and keywords for a systematic literature review on learnability of data-driven models for CPSs. 22
Table III Excluded concepts and keywords for a systematic literature review on learnability of data-driven models for CPSs. 22
Table IV Parameters for the symbolic regression algorithm. 39
Table V Overview of the examples and their appearances this thesis. 40
Table VI Learning results with 100 traces of length 10. The distance covered by a `move` command, the time needed to learn the automaton and the number of queries needed to learn the automaton are shown. 59
Table VII Number of generated test cases for state coverage $\mathcal{O}_{SC}$ and transition coverage $\mathcal{O}_{TC}$ for different input abstractions with different numbers of input symbols $|I|$ and output symbols $|O|$. 65
Table VIII Empirical results for the analysis of automata forests. The presented metric is the average μ of the number of traces from the test set

of size 1000 that were correctly translated by the learned model, the overall execution time t in seconds and the p-value of the Welch's t-test for the hypothesis that the samples of the automata forests have a larger mean value, are given. The columns show the results for the traditional RPNI, the automata forest with performance metrics (Forest $_{\mathrm{PM}}$ count), where the count of correctly translated traces is maximized, the automata forest with performance metrics ($\mathrm{Forest}_{\mathrm{PM}}$ Hamming), where the Hamming distance is minimized, and for the automata forest with majority vote ($\mathrm{Forest}_{\mathrm{MV}}$). 72

Table IX Prediction accuracy of decision tree models (T) and neural networks (NN) on complete learning sets for different n. The number of nodes in the decision tree is given as $|V|$ and the number of neurons in the neural network is given by the column v_{NN}. The number of impure leaves is given as $|V_a|$. The learning time in seconds is given as t_{learn}. . 90

Table X Prediction accuracy in terms of false positive and true positive predictions of decision tree models (T) and neural networks (NN) and automaton models (A) on practical learning sets for different n. The number of nodes in the decision tree is given as $|V|$ and the number of neurons in the neural network is given by the column v_{NN}. The number of ambiguous leaves is given as $|V_a|$. The learning time in seconds is given as t_{learn}. 91

Table XI Prediction accuracy of the regression tree model in terms of mean absolute error and maximum absolute error for different n. The number of nodes in the regression tree is given as $|V|$ and the number of neurons in the neural network is given by v_{NN}. The learning time in seconds is given as t_{learn}. 95

Table XII Parameters for the FaMoS algorithm. 125

Table XIII Parameterization of FaMoS for all examples. 125

Table XIV Parameterization of HAutLearn for all examples. 125

Table XV Learned flow functions of FaMoS and HAutLearn. A '-' indicates a timeout of 5 hours for the learning process. 125

Table XVI Evaluation metrics of FaMoS (FaM) and HAutLearn (HAut) consisting of the segmentation objective function Ω_{seg}, the grouping objective function Ω_{group}, and the MSE of the normalized predicted traces to the ground truth traces. 126

Table XVII Execution times of the individual steps of FaMoS and HAutLearn in seconds. The segmentation and mode characterization is the same for all methods. Thus, only the time of the FaMoS run is given. The extraction method is the same for both variants of FaMoS, thus, only the time for the FaMoS run is shown. A timeout of 5 hours is set for each example; '-' indicates that a timeout happened. 128

Table XVIII Parameters for system identification with SR for all examples. 138
Table XIX Learned expressions per group for the examples. 139
Table XX Accuracy of the hybrid decision tree in the segmentation step in terms of segmentation metric Ω_{seg}, the grouping step in terms of grouping metric Ω_{group}, and the MSE of the predicted traces on the evaluation set. 140
Table XXI Execution times of the individual steps of model learning with symbolic regression in seconds. 142
Table XXII Properties of the learning approaches in this thesis: Active Automata Learning (AAL), Passive Automata Learning (PAL), Automata Forests (AF), Decision Tree Learning (DTL), Regression Tree Learning (RTL), Fast Modeling for Cyber-Physical Systems (FaMoS), and Symbolic regression for hybrid system identification (SR). 155
Table XXIII Usage of examples in the evaluation of the different learning approaches: Active Automata Learning (AAL), Passive Automata Learning (PAL), Automata Forests (AF), Decision Tree Learning (DTL), Regression Tree Learning (RTL), Fast Modeling for Cyber-Physical Systems (FaMoS), and Symbolic regression for hybrid system identification (SR). A '✓' indicates that the approach is evaluated with the example. A '+' indicates that a variant of the approach is evaluated on the example. The color coding shows the performance of the model, where a green cell indicates good performance, yellow indicates moderate performance, and orange indicates poor performance. 157
Table XXIV Accuracy (Acc.) in % and learning time t_{learn} in seconds for the discrete models, learned with passive automata learning, automata forests and DTL. 160
Table XXV Mean square error (MSE) of the continuous approaches on the evaluation trace of the examples and learning time t_{learn} in seconds. . 162

Index of Listings

Listing 1 RPNI algorithm for passive learning of a DFA from positive and negative observations [79]. The algorithm uses the positive observations to create a tree structure and successively merges the tree nodes into an automaton considering the negative observations. The result is a DFA that accepts the positive observations and rejects the negative observations. 32
Listing 2 Generic algorithm for building an automata forest. The steps are the creation of subsets and the learning of automata from these subsets. . . . 67
Listing 3 Classification of a trace $\boldsymbol{s}$ using a decision tree model with language-based binary classification function d. 82

Listing 4 Function to select the next leaf to visit. The input is the set V_{next} of reachable leaves, the set R_L of reached leaves, the decision tree $\mathcal{T}$, the system $\mathcal{S}$, the current feature vector $\boldsymbol{f}$, and the parameter $s_{\text{heuristic}}$ to select the heuristic. 100
Listing 5 Function to determine the feature vector after the application of the input trace $\boldsymbol{i}$. This is done by a consecutive update of the feature vector $\boldsymbol{f}$ querying either the decision tree model $\mathcal{T}$ or the system $\mathcal{S}$. 101
Listing 6 Algorithm for automatic test generation with a decision tree model $\mathcal{T}$ for the system $\mathcal{S}$ using the heuristic defined by $s_{\text{heuristic}}$. Reachable leaves that are not yet visited are identified to optimize the leaf coverage metric. . 101
Listing 7 Inference of the hybrid decision tree. The relevant flow function is selected using the decision tree and then evaluated for the sample $\boldsymbol{x}$. . 115
Listing 8 Grouping process in FaMoS: first, the derivative κ_g, which is used for the grouping, is determined. Then, all possible pairs of segments are created. For every pair and every variable, DTW-inspired comparisons are executed. The results are stored in a comparison matrix $\boldsymbol{S}$ for each variable. Groups per variable are combined to global groups. An optional LMI-based refinement of the grouping is performed. 118
Listing 9 Detection of transition points with SR. The algorithm considers a window of the trace and extends this window until the segmentation criterion is not fulfilled anymore. A transition point is detected, and the window is stored as a segment. 131
Listing 10 The segmentation criterion of the segmentation step compares the current error e with the stagnation threshold ξ_S and the previous error e_{prev}. If the current error is smaller than the stagnation threshold or smaller than the previous error, the segmentation criterion is fulfilled. 131
Listing 11 Grouping of segments with SR. The algorithm iterates over all segments and groups them with the help of symbolic regression. The segments are grouped if the groupingCriterion() is fulfilled. If no group is found, a new group is created. 134
Listing 12 The grouping criterion of the grouping step learns expressions for the conjunction of the current segment $\diamond\, \boldsymbol{s}$ and the current group g and evaluates whether the error of the learned expression on the conjunction is smaller than the error of the learned expression on the group alone multiplied by a relaxation parameter ξ_G. 136