Dipl.-Ing. Edi Muškardin, BSc.

Modelling and Analysis of Machine Learning Systems with Automata Learning

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Main Supervisor
Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Bernhard Aichernig
Institute of Software Technology (IST)

Co-Supervisors
Dr.techn. Dipl.-Ing. Martin Tappler
Institute of Computer Engineering, TU Wien

Dr.techn. Dipl.-Ing. Ingo Pill
Silicon Austria Labs (SAL)

External Reviewer and Examiner
Prof. Dr. Falk Howar
Department of Computer Science, TU Dortmund University

Graz, August 27, 2024

Dipl.-Ing. Edi Muškardin, BSc.

Modellierung und Analyse von maschinellen Lernsystemen mit Automatenlernen

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften

eingereicht an der

Technischen Universität Graz

Hauptbetreuer
Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Bernhard Aichernig
Institut für Softwaretechnologie (IST)

Co-Betreuer
Dr.techn. Dipl.-Ing. Martin Tappler
Institut für Computer Engineering, Technische Universität Wien

Dr.techn. Dipl.-Ing. Ingo Pill
Silicon Austria Labs (SAL)

Externer Gutachter und Prüfer
Prof. Dr. Falk Howar
Fakultät für Informatik, Technische Universität Dortmund

Diese Arbeit ist in englischer Sprache verfasst.

Abstract

Machine learning affects our daily lives without us even realizing it. Machine learning-based systems drive cars, trade stocks, make medical diagnoses, and moderate content, among others. Many of these systems were not explicitly programmed, but have automatically learned their behavior from data. And while their performance often surpasses that of human experts, a fundamental question remains: "How and why do they make their decisions?" Or when viewed from a scientific perspective, we might ask: "What is the model behind their reasoning?"

In this thesis, we approach these questions with the help of automata learning. Automata learning, otherwise known as model inference, is a set of techniques that automatically model the behavior of stateful systems, either by interaction or from provided data. Learned models capture the input-output behavior of a system under learning and encode it as a formal model, often as a finite automaton.

The work presented in this thesis discusses the applications of automata learning in the domain of machine learning from a practical perspective. Presented contributions could be divided into three categories: (1) advancements in automata learning, (2) automated modeling of RNNs with automata learning, and (3) a study of possible combinations of automata learning and reinforcement learning (RL). We start by designing and developing AALPY, an automata learning library that enables a seamless integration of automata learning and machine learning. We extend it with novel automata learning algorithms and throughout the thesis we demonstrate its capabilities in the machine learning domain. We then use automata learning to mine behavioral models from RNNs trained on regular languages. In this context, we also reason about the appropriate abstraction level that would enable the scaling of the proposed techniques to RNNs trained on higher-level domains such as natural language. Finally, we show how automata learning can be used in combination with reinforcement learning. We start by demonstrating how automata learning can extend reinforcement learning either by enabling decision-making in the the presence of partial observability or by enabling safe training of RL agents via shielding over learned models. Finally, we develop a method for automated modeling of complex environments with continuous observations and stochastic dynamics. We have evaluated this method on several RL benchmarks and have shown how it can be used for differential testing of deep RL agents.

Keywords: Automata Learning, Machine Learning, Automated Modeling, Recurrent Neural Networks, Reinforcement Learning, Software Testing.

Kurzfassung

Maschinelles Lernen beeinflusst unser tägliches Leben, ohne dass wir uns dessen bewusst sind. Systeme, die auf maschinellem Lernen basieren, fahren Autos, handeln mit Aktien, stellen medizinische Diagnosen und moderieren Inhalte, um nur einige Beispiele zu nennen. Viele dieser Systeme wurden nicht explizit programmiert, sondern haben ihr Verhalten automatisch aus Daten gelernt. Und obwohl ihre Leistungen oft die von menschlichen Experten übertreffen, bleibt eine grundlegende Frage bestehen: "Wie und warum treffen sie ihre Entscheidungen?" Oder aus wissenschaftlicher Sicht könnte man fragen: "Was ist das Modell hinter Ihrem Handeln?"

In dieser Arbeit nähern wir uns diesen Fragen mit Hilfe des Automatenlernens. Automatenlernen, auch bekannt als Modellinferenz, umfasst eine Reihe von Techniken, die automatisch das Verhalten von zustandsabhängigen Systemen modellieren, entweder durch Interaktion oder aus bereitgestellten Daten. Gelernte Modelle erfassen das Input-Output-Verhalten eines zu lernenden Systems und kodieren es als formales Modell, häufig in Form eines endlichen Automaten.

Die in dieser Dissertation vorgestellte Arbeit diskutiert Anwendungen des Automatenlernens im Bereich des maschinellen Lernens aus einer praktischen Perspektive. Die vorgestellten Beiträge können in drei Kategorien eingeteilt werden: (1) Fortschritte im Automatenlernen, (2) automatisierte Modellierung von RNNs mittels Automatenlernen und (3) Studien über mögliche Kombinationen von Automatenlernen und Verstärkungslernen (VL). Wir beginnen mit dem Entwurf und der Entwicklung von AALPY, einer Automatenlernbibliothek, die eine nahtlose Integration von Automatenlernen und maschinellem Lernen ermöglicht. Wir erweitern sie um neuartige Algorithmen für das Automatenlernen und demonstrieren in dieser Arbeit ihre Funktionalitäten im Bereich des maschinellen Lernens. Anschließend setzen wir Automatenlernen ein, um aus RNNs, die auf regulären Sprachen trainiert wurden, Verhaltensmodelle zu erstellen. In diesem Zusammenhang betrachten wir geeignete Abstraktionen, die eine Skalierung der vorgeschlagenen Techniken auf RNNs ermöglichen würden, die auf Sprachen höheren Ebenen wie natürlicher Sprache trainiert werden. Schließlich zeigen wir, wie Automatenlernen in Kombination mit Verstärkungslernen verwendet werden kann. Wir beginnen mit der Demonstration, wie Automatenlernen das Verstärkungslernen erweitern kann, indem es entweder die Entscheidungsfindung bei partieller Beobachtbarkeit oder ein sicheres Training von VL-Agenten durch Abschirmung über gelernte Modelle ermöglicht. Schließlich entwickeln wir eine Methode zur automatischen Modellierung komplexer Umgebungen mit kontinuierlichen Beobachtungen und stochastischer Dynamik. Wir haben diese Methode an mehreren VL-Benchmarks evaluiert und gezeigt, wie sie für differenzielle Tests von Deep-VL-Agenten verwendet werden kann.

Schlagworte: Automatenlernen, Maschinelles Lernen, Automatisierte Modellierung, Rekurrente Neuronale Netze, Verstärkungslernen, Software Testen.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

. . . . .	. . . . .
place, date	(signature)

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

. . . . .	. . . . .
Ort, Datum	(Unterschrift)

Acknowledgements

The road leading to this thesis was long, but not lonely. I express my deepest thanks to everybody who helped me along my journey, both academically and personally.

I thank my supervisor Bernhard Aichering. During my master's studies he introduced me to the topic of automata learning in one of his lectures, and ever since then we have been working together on all aspects related to this field of research. I am especially grateful for the given academic opportunities, his guidance, and all the discussions we had throughout the years. Aside from being my supervisor, Bernhard has provided helpful advice to my newly established family which I appreciate.

I am also grateful to my co-supervisors Martin Tappler and Ingo Pill. Ingo has helped me make decisions that led to me continuing my studies and was always supportive of my pursuits. Martin has affected all aspects of my work, both by providing invaluable advice, guidance, and most importantly, by being a true friend. Martin's wholesome personality greatly contributed to my student life at TU Graz, and even more so to my life in Austria. Today, due to such friendships I can call Graz my home.

I thank all my colleagues at Silicon Austria Labs for all the laughs and good times we had throughout the years. Special thanks go to Andrew Wilson, Victoria Fill, Chiara Gei, Silvester Sabathiel, Florian Lorber, Jakub Breier, Ambily Suresh, Gleb Radchenko, Manuel Freiberger, Willibald Krenn, and Monika Stipsitz for being there and bearing with me. Special mention goes to Hèlios Sanchis Alepuz, a worthy adversary in all intellectual debates, such as whether the number zero exists and the "alleged existence" of gravity.

At the TU Graz, I have worked with and created many beautiful memories with Martin Tappler, Andrea Pferscher, Benjamin von Berg, Felix Wallner, and Bettina Könighofer. I have barged into your office one too many times and interrupted your work, but I hope that you have enjoyed my company as much as I did yours. All of you are great researchers and I wish you a long and prosperous academic career.

I also express my gratitude to Falk Howar for serving as my external examiner. In addition, I am thankful to him and his team at TU Dortmund for their continuous contributions to the automata learning landscape. I fell in love with automata learning ever since I learned and visualized the model of a car alarm system with LearnLib.

Finally, I thank everybody in DES-Lab, especially Willibald Krenn and Bernhard Aichernig, as they have managed the project that has enabled me to perform fundamental research. My thesis and all presented work have been supported by the "University SAL Labs" initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic-based systems.

Few words in Croatian for my family: Posebno hvala mojim roditeljima Damiru i Suzi za predivno djetinstvo i podršku, sestri Andrei za puno ljepih uspomena, i noni Paoli za sve. Hvala Vam od srca.

As for my wife, I cannot express with words the influence Gita had on my life, so let's try like this: ♡

Edi Muškardin
Graz, Austria, August 27, 2024

Table of Contents

List of Figures

4 ACTIVE MODEL LEARNING OF STOCHASTIC REACTIVE SYSTEMS

5 BLACK-BOX EXTRACTION OF FINITE STATE MODELS FROM RNNS

6 ON THE RELATIONSHIP BETWEEN SEMANTIC STRUCTURES AND RNN HIDDEN-STATE VECTORS

7 EXTENDING REINFORCEMENT LEARNING WITH AUTOMATA LEARNING

8 AUTOMATA LEARNING MEETS SHIELDING

9 MODELLING AND TESTING OF DEEP-RL AGENTS IN CONTINUOUS STOCHASTIC ENVIRONMENTS

List of Tables

List of Algorithms

List of Acronyms

AI artificial intelligence.

ANN artificial neural network.

BMDP Belief MDP.

DFA deterministic finite automata.

DRL deep reinforcement learning.

DT decision trees.

FM formal methods.

FSM finite-state machines.

GRU Gated recurrent units.

GSM generalized state merging.

IOFPT input-output frequency prefix tree.

IOPT input-output prefix tree.

LDA linear discriminant analysis.

LR logistic regression.

LSTM Long short-term memory networks.

MAT minimally Adequate Teacher.

MDP Markov decision process.

ML machine learning.

MSE mean squared error.

ONFSM observable non-deterministic finite-state machine.

PDA pushdown automaton.

POMDP partially observable Markov decision process.

RL reinforcement learning.

RNN recurrent neural network.

SMM stochastic Mealy machine.

SUL system under learning.

VPDA deterministic visibly pushdown automata.

1 Introduction

1.1 Motivation

Machine learning-based systems make decisions that affect our daily lives and shape the future. Machine learning (ML) can be used to drive cars [18, 281], play computer games [25, 224], create new content [37, 84], help experts regardless of their field of study [62, 211], and much more. The growth of machine learning and its usefulness cannot be overstated. While machine learning has had exponential growth both in academic and industrial applications in recent years [115, 201], the broader public was first introduced to machine learning through ChatGPT [188]. Since its release in late 2022, it has been a disruptive force in workspace, education, and content creation [133, 211].

And while machine learning can (and does) make decisions that affect human lives, a fundamental question of trust in machine learning-based systems remains unsolved. ML-based systems can make decisions that surpass an expert in a given field (e.g. in medical diagnosis [132] and in the game of Go [224]), but questions of *how* and *why* these decisions were made remain unanswered. Various approaches towards the explainability of ML-based systems have been made [147, 214, 259, 284], however, the progress in explainability is still lagging behind the advancements in ML. This fundamental lack of explainability, as well as consequences of potential misuse, are at the core of the European Union AI Act [55].

> *High-risk AI systems should be designed and developed in such a way that natural persons can oversee their functioning.*
>
> – European Union AI Act

Machine learning-based systems are trained to find patterns in data [28]. They depend on the quantity and quality of data, the training process, and on the parameterization of their architecture. After they are trained they can perform a specific task, and even find novel solutions to existing problems [85]. However, with ML-based systems we are left without a concept that is inherent to science: an explainable model. Models are at the heart of science and engineering. They help us understand complex physical and software systems and can be used to make predictions about a system's behavior. Through the scientific process, humans have inferred models that are consistent with observations of the world and only rely on several axioms. In a twist of irony, as a consequence of scientific progress that is solely based on explainable models, scientists have developed systems that can surpass their expertise *without*

an explicit model. In our work, we make contributions towards the explainability of machine learning-based systems.

More concretely, we attempt to formalize the behavior of ML-based systems with formal methods (FM). Formal methods are a set of techniques that formalize the behavior of engineered systems through mathematical rigor. Formal methods offer modeling formalisms that can be used to specify systems behavior at several levels of abstraction, from formal specification and verification to implementation. A formal model of a system can be used to prove the correctness of the system with respect to some properties. This differs from the purely classical empirical view of testing, where software systems are subjected to many hand-written and automatically generated test cases. Formal methods are used to verify many engineered systems, most notably to verify the correctness of hardware [36] and to model safety-critical systems [129, 181]. A noteworthy example from the industry: TLA+ has been used at Amazon to optimize their development process, formally prove the correctness of new algorithms and data structures, and to find bugs in their existing products [181].

Formal methods come with a cost: manual creation of a model. This step alone can prevent the use of formal methods, as model development requires many man-hours and a trained workforce [126]. In addition, the model designer has to have a deep understanding of the system under modeling. While ML-based systems could be used to learn the behavior of a system, but they cannot be used to reason about the system structure itself. However, formal methods provide us with techniques that can infer system behavior and provide us with a formal model of the system's structure. This techniques are called automata learning.

Automata learning automatically infers the behavior of the system, either by interaction or from provided system traces. Automata learning was first researched by Moore [157], Gold [83], and Angluin [14]. They formally examined "the limits of learnability", that is, what formal structures can be induced from given symbolic data. Since then, these formal methods of system inference have been used to automatically model [192] and test [5] engineered black-box systems. In this thesis, we show how automata learning can be used to automatically model machine-learned systems. In our work, formal methods meet artificial intelligence, or more precisely, *automata learning meets machine learning*.

1.2 Automata Learning

Automata learning, also known as model learning [5], automata inference [215], or grammatical inference [58], is a branch of formal methods that is concerned with the automatic identification of system behavior and modeling it as automata. Automata are mathematical structures that describe the stateful behavior of some system, and how it produces outputs based on given inputs. As such, automata learning is useful in dealing with stateful systems that can be modeled as finite state automata (with finite discrete input and output alphabets). In the context of this thesis, we consider automata learning algorithms that model system behavior as deterministic and stochastic regular languages. However, automata learning has also been studied for other formal structures, such as timed automata [236] and register automata [110]. Automata learning methods can be further divided into *active* and *passive* learning.

Active learning methods like L^* [14] and TTT [109] learn the model of system under learning (SUL) by active interaction with the SUL. They iteratively build the model of the SUL by asking two types of queries: *input* queries and *equivelance* queries. Input queries are a sequence of inputs for which an output is observed when they are executed on the SUL. For example, when the input query *(connect, subscribe, publish)* is executed on some system, outputs *(conn_succ, subs_succ, msg_pub)* are observed. Once sufficient information is gathered by asking input queries, the learning algorithm constructs a hypothesis (a learned model of the system) that is consistent with observed outputs. This model is then passed to the equivalence oracle. The equivalence oracle attempts to prove the equivalence between the learned model and SUL through hypothesis falsification, that is it attempts to provide a counterexample between the inferred hypothesis and the SUL. A counterexample is an input sequence that displays different behavior on the SUL and on the inferred hypothesis. If the counterexample is found, it is processed by

the learning algorithm (by asking new input queries) and the hypothesis is extended. However, if the equivalence oracle is unable to find any more counterexamples, the active learning algorithm halts and returns the final hypothesis as a learned model.

On the other hand, *passive automaton learning* algorithms learn from provided data, without interaction with the SUL. Therefore, the quality of the learned model is limited by the available data, whereas in active learning new data can be obtained by posing input queries. Notable passive learning algorithms are RPNI [187] and ALERGIA [41, 144]. The former is used to infer deterministic finite-state models from the data, while the latter infers stochastic models. Both algorithms share the working principle called state merging. The algorithms start by creating an input-output tree containing all provided input-output sequences, and the nodes in the tree are merged if they display the same future behavior. Passive learning results in an automaton that is consistent with the provided data. While active learning algorithms provide stronger guarantees about the accuracy of the learned model (as they can access more data when needed), passive learning algorithms are useful in scenarios where interaction with the system is not possible, or where the interaction with the system might be costly.

Automata learning in the research context. In the thesis, we show how automata learning can be used as a black-box modeling technique, with the focus on the modeling of machine learning systems. Therefore, we developed AALPY, a state-of-the-art automata learning library written in Python. It serves as a comprehensive solution for automata learning and contains efficient implementations of existing, as well as novel algorithms. Throughout the thesis, we show how automata learning can be applied in the domain of machine learning, and discuss potentials and challenges.

1.3 Machine Learning

Machine learning is a subfield of artificial intelligence (AI) that studies statistical algorithms that learn from data, and can generalize to previously unseen data [28]. In the last decade, the increase in computational power, as well as the availability of vast datasets, has led to the successful application of machine learning in various domains [18, 115, 133, 211]. These successes led to increased interest in the domain of machine learning, which resulted in the improvement of existing ML architectures and the development of new ML architectures, like generative adversarial networks [84] and transformers [254].

The field of machine learning can be categorized into three categories: unsupervised learning, supervised learning, and reinforcement learning.

Unsupervised learning is the type of machine learning that learns from unlabeled data. It is used to identify properties in continuous data without any human supervision. The most prominent subcategories of unsupervised learning are dimensionality reduction and clustering. Dimensionality reduction techniques aim to reduce the dimensionality of continuous data. Reduced representation of high-dimensional data helps with future extraction/engineering, and visualization of the data, and can increase the computational efficiency of other machine learning methods. Prominent dimensionality reduction techniques are PCA [227], t-SNE [253], and autoencoders [20]. On the other hand, clustering techniques identify similar points in the dataset (according to some criterion/distance) [70]. Clustering is used to find patterns in the data and can be seen as an unsupervised classification technique. Among others, it can be used for image/text classification and analysis, anomaly detection, and to find similar behavior in recommender systems [70]. The most prominent clustering method is k-means [90], which splits the dataset into exactly k categories/clusters.

Supervised learning learns statistical relationships between continuous input data and labeled output data. Output data can be discrete (classification) or continuous (regression). In this thesis, we mostly consider two types of supervised learning: regression models and artificial neural networks (ANNs). Regression models (linear/logistic) are used to model the relationship between dependent and independent variables [29]. When using a regression model, one has to assume a functional form (e.g. data can be modeled with a quadratic function). After the regression models are fitted they can be used to

make predictions, as well as to reason about the dependencies between variables. On the other hand, neural networks relax the assumptions about functional relationships between inputs and outputs and can find input-output relationships even in non-linear systems. Special types of neural network architectures, namely recurrent neural networks (RNNs) and transformers, can be used to analyze and process sequential data [47, 67, 95, 254]. However, the power or ANNs comes with a cost: lack of explainability [44, 174].

Compared to supervised and unsupervised learning, which learns from data, reinforcement learning (RL) learns by interaction with the system. The goal of reinforcement learning is to find a solution for a given task, called a policy. During the training process, RL algorithms make decisions in the environment based on the observed rewards and past experiences. In essence, RL algorithms aim to find an (optimal) solution to a problem so that the policy maximizes the reward. RL algorithms like Q-learning [269] can be used to find an optimal policy in discrete observation space environments, whereas a combination of basic RL principles with function approximators, namely neural networks, has been used to solve complex problems, such as games of Go [224] and DOTA [25].

Machine learning in the research context. In the scope of the thesis, we examine the applicability of automata learning in the domain of machine learning. Since many machine learning systems work on continuous data, compared to automata learning which works on symbolic data, we also recognized a need to use machine learning to aid automata learning. We show how ML techniques can enable automata learning in the continuous domain. More precisely, we have used methods from unsupervised machine learning (e.g. clustering) to enable the modeling of supervised machine learning techniques (deep neural networks) and of reinforcement learning.

1.4 Research Context and Thesis Scope

1.4.1 Research Project

The work that resulted in this thesis was performed in the TU Graz - SAL Dependable Embedded Systems Lab (DES Lab), a research collaboration between Silicon Austria Labs (SAL) and Graz University of Technology (TU Graz). DES Lab focuses on fundamental research in order to provide the dependability of modern computer-based systems, with a special focus on intelligent and networked systems. The DES Lab has conducted interdisciplinary basic research in the areas of software testing, automatic modeling, verification, machine learning, optimization, and hardware-related software development.

The first phase of the DES Lab project was active from October 2020 until the end of 2023. After the successful evaluation of the project, an extension phase was granted.

Note on Collaborations

All work considered in the thesis was done in the scope of the DES Lab project. Therefore, all papers were a collaboration between Graz University of Technology and Silicon Austria Labs. As a consequence, all considered papers include a combination of the following co-authors:

- Bernhard K. Aichernig: professor at the Institute of Software Technology, Graz University of Technology. He acted as a supervisor of my thesis and as a chairman of the DES Lab.
- Ingo Pill: a staff scientist at Silicon Austria Labs and a member of the DES Lab.
- Martin Tappler: former postdoc at Bernhard Aichernig's group and a member of the DES Lab, currently a postdoc at Vienna University of Technology

Collaboration with this core team was fundamental to all papers that contributed to the thesis. Other co-contributors are explicitly mentioned in Sect. 1.5.

1.4.2 Problem Statement

Research Scope

This thesis examines the applicability of automata learning in the domain of machine learning. When defining the research scope, we examined the machine learning sub-fields in which automata learning could be feasibly applied. Given that automata learning can only be applied for automatic modeling and analysis of stateful systems, we limited our scope to:

- **Recurrent Neural Networks**: Whereas classic feed-forward neural networks can be used on static data (e.g. image classification), recurrent neural networks can model (high-dimensional) sequential data (e.g. audio streams). RNNs are inherently stateful, that is, the output of the RNN depends on the input and the current state, which is encoded in its hidden state vectors. RNNs' statefulness makes them a perfect candidate for analysis, as we can apply automata learning to model their stateful behavior, and also to examine the properties of hidden-state vectors.
- **Reinforcement Learning**: In reinforcement learning, an agent makes decisions in a (stochastic) environment based on the environment's reward structure. Reinforcement learning is inherently stateful as the agent's decisions depend on the current state of the environment, and the decision-making process is guided by expected future rewards. As such, automata learning can be applied to model reinforcement learning environments themselves, or to model agents' behavior in the environment.

After defining these fields of interest, we observed a need for an automata learning framework suitable for modeling machine learning systems. In the related work, we observed that automata learning algorithms were implemented on a per-need-basis [64, 145, 271], and their implementations were hardly transferable. In addition, we noticed the need for advancements in the modeling of stochastic systems, as stochastic behavior is often inherent to problems tackled by machine learning, as it arises from epistemic (lack of knowledge about the system) or aleatoric (inherent randomness or variability of the system) uncertainty [103].

Research Questions

We formalized the areas of interest identified in the previous section through three research questions. Firstly, we examine the capabilities of automata learning from a practical perspective. To that end, we design and develop an automata learning framework that can be used to model various black-box systems. To enable efficient automata learning, we improved upon existing algorithms and designed novel algorithms. We put special focus on the design and development of efficient stochastic automata learning algorithms since they can model uncertainty, which is often inherent to machine learning tasks. These objectives are summarized in **RQ 1**.

- **RQ 1** How can automata learning techniques be seamlessly employed in the machine learning domain?
  - **RQ 1.1** How can existing automata learning approaches be improved to enable efficient automata learning?
  - **RQ 1.2** How can we improve upon the state of the art in modeling of stochastic systems?

Once an automata learning framework is established, we examine how can we apply automata learning to model and analyze RNNs. Firstly, we examine existing model-extraction techniques for RNNs trained to recognize regular languages and improve upon these. After the model extraction process has been established for RNNs trained on discrete data, we analyze the appropriate abstractions that can be employed to enable model extractions from an RNN trained on continuous data. Proposed techniques and findings will be discussed in **RQ 2**.

- **RQ 2** How can automata learning be used to model recurrent neural networks?
  - **RQ 2.1** Can automata learning be used to efficiently extract behavioral models from RNNs?
  - **RQ 2.2** What is an appropriate abstraction for the modeling of RNNs?

Finally, in **RQ 3** we examine how can automata learning methods be combined with reinforcement learning. We start our inquiry by extending tabular reinforcement learning techniques with knowledge inferred by automata learning, thus enabling reinforcement learning in partially observable domains, as well as safe reinforcement learning via shielding. In addition, we examine how automata learning can be used to model continuous stochastic environments, and whether models inferred through automata learning can be used for other purposes, such as testing of deep reinforcement learning agents.

- **RQ 3** How can automata learning be combined with reinforcement learning?
  - **RQ 3.1** Can automata learning help reinforcement learning?
  - **RQ 3.2** Can automata learning be used to model the dynamics of continuous stochastic environments?
  - **RQ 3.3** Can automata learning enable the testing of deep reinforcement learning agents?

1.5 Contributions and Publications

1.5.1 Thesis Contributions

- We developed AALPY [164, 166], a state-of-the-art automata learning library, which contains optimized versions of existing automata learning algorithms, as well as implementations of newly developed algorithms. AALPY contains many auxiliary functionalities: generation of random automata, multiple model-based testing strategies, and interfaces to model checkers, among others.
- We have demonstrated AALPY's applicability on many real-world examples. While the thesis focuses on the machine learning domain [167, 169, 241, 242], we have also applied our library in other domains, e.g. to learn network protocols [35, 198].
- We have developed a new algorithm for efficient inference of stochastic regular languages [171, 238]. The algorithm outperforms the accuracy of the state-of-the-art while requiring fewer interactions with the system under learning.
- We proposed a model extraction method for RNNs trained on regular languages [167]. This method was further extended to be able to learn models trained on regression tasks [168].
- With our method for model extraction from RNNs we won the TAYSIR competition [69]. The goal of the competition was the development of methods that enable the extraction of simple and interpretable models from transformers and RNNs.
- We have conducted a comprehensive empirical analysis of the assumption that RNN's hidden state vectors form clusters that correspond to semantic structures [172].
- We have extended tabular reinforcement learning algorithms with the knowledge inferred by automata learning, enabling reinforcement learning in partially observable environments [169].
- We have developed a framework that combines automata learning and shielding to enable safe reinforcement learning in partially observable environments [241].

- We developed CASTLE, the first automata learning method for modeling stochastic continuous environments [242]. CASTLE can learn finite-state representations of complex environments. The accuracy of learned models was evaluated on stochastic reinforcement learning environments with continuous state representation.
- We proposed and evaluated a method for differential testing of deep RL agents [242]. The method is based on our algorithm from modeling of stochastic continuous environments.
- We put special focus on ease-of-use and reproducibility of our results. We made all of our tools and research open-source with permissive licenses:
  - AALPY: an automata learning library [173]
  - Framework for extraction of models from RNNs [161]
  - Framework used to participate in the TAYSIR competition [163]
  - Framework for analysis of RNNs hidden state vectors [162]
  - Framework for RL in partially observable environments [160]
  - Framework that combines model learning and shielding for RL [158]
  - Method, experimental setup, and application of automata learning in stochastic continuous environments [159]

1.5.2 Main Publications

The statutes of the Doctoral School of Computer Science at Graz University of Technology require that a doctoral dissertation includes an annotated list of publications, highlighting individual contributions in these publications. This thesis considers in total 10 publications: 7 conference papers, 2 journal papers, and 1 paper under submission.

1. ATVA 2021: "AALPY*: an Active Automata Learning Library*" [164]. This paper presents AALPY, our active automata learning library. AALPY was developed to address the lack of automata learning libraries or frameworks that could be used in the Python ecosystem. AALPY contains efficient implementations of deterministic, non-deterministic, and stochastic automata learning algorithms, as well as a dozen testing strategies that can be used to answer equivalence queries. I implemented the majority of AALPY, and described AALPY's structure, interfaces, and auxiliary functions in the paper. I presented the paper virtually at the ATVA conference in 2021.

2. SEFM 2022: "*Active Model Learning of Stochastic Reactive Systems*" [238]. This paper presents a method for active learning of stochastic Mealy machines, as well as various adaptations of the L^*_{MDP} algorithm. These adaptations greatly improve the efficiency of active learning of stochastic reactive systems. I introduced the stochastic Mealy machine formalism to automata learning, developed a new stopping criterion, adapted counterexample processing strategies to a stochastic setting, and redesigned the majority of the original algorithm. The paper was presented at SEFM 2021, where it **won the best paper award**.

3. iFM 2022: "*Learning Finite State Models from Recurrent Neural Networks*" [167]. In this paper, we used active automata learning to extract models from RNNs trained to recognize regular languages. Compared to competing approaches, our method produces more accurate models with no additional overhead. I created a framework for model extraction and evaluation, as well as developed multiple model-based testing techniques that were used to answer the equivalence query. I actively contributed to the writing of every part of the paper, with a focus on the method and evaluation sections. I presented the paper virtually at iFM 2022.

4. ISOLA 2022: "*Automata Learning Meets Shielding*" [241]. This paper is a collaboration with Bettina Könighofer Stefan Pranger, Roderick Bloem, and Kim G. Larsen. In the paper, we combined automata learning and shielding to enable safe RL in partially observable environments. I designed and implemented the algorithm and parts of the evaluation framework, and have contributed to the writing of the method and evaluation sections. The paper was presented at ISOLA 2022.

5. Innovations in Systems and Software Engineering 2022: A NASA Journal "*AALpy: an Active Automata Learning Library (Extended Version)*" [166]. In the journal version of the AALPY paper [164], we extend all parts of the original paper and add additional descriptions of AALPY's interface and structure, available equivalence oracles, and describe several applications. I have contributed to every aspect of the extension.

6. ICGI 2023: "*Testing-based Black-box Extraction of Simple Models from RNNs and Transformers*" [168]. This paper builds upon our work on the extraction of models from RNNs. The paper outlines the model-extraction technique that was used to participate in the TAYSIR competition [69]. In the paper, we also introduced a technique that can be used to learn finite state models of RNNs trained on regression. I designed the mapper used for learning regression models, implemented the framework that was used in the competition, and was the main author of the paper. The proposed model extraction method **won the TAYSIR competition**.

7. iFM 2023: "*Reinforcement learning under partial observability guided by learned environment model*" [169]. In this paper we show how models learned with passive automata learning can aid RL algorithms in the partially observable setting. The method combines automata learning and reinforcement learning. The reinforcement learning agent explores the environment, and models learned with automata learning are used to provide additional knowledge to the RL agent. I have designed and implemented parts of the algorithm, the evaluation framework, and was an active contributor in the whole paper writing process, with a focus on the method and evaluation sections. I have presented the paper at iFM2023.

8. SoSyM 2024: "*Active Model Learning of Stochastic Reactive Systems (Extended Version)*" [171]. In the extended version of our SEFM paper on stochastic model learning [238], we extend all parts of the paper, add additional benchmarks (both random models and derived from real-world examples), examine the influence of different counterexample processing strategies on the algorithm, and we examine the accuracy of learned models by performing statistical model checking. I have contributed to every aspect of the extension.

9. ICST 2024: "*Learning Environment Models with Continuous Stochastic Dynamics – with an Application to Deep RL Testing*" [242]. In this paper, which is joint work with Bettina Könighofer, we designed and developed a novel method of modeling potentially infinite continuous environments with finite-state models. Our method, named CASTLE, uses dimensionality reduction and clustering to group similar observations and uses these equivalence groups as an abstraction mechanism that enables automata learning. We evaluate our approach on existing RL benchmarks and show how it can be generalized to other applications, such as testing of deep RL agents. I was an active participant in the design of the algorithm and its application to differential testing. I have written the parts of the evaluation and differential testing sections. In addition, I have implemented a major part of the algorithm, part of the evaluation framework, and the whole differential testing framework. Our paper was presented at ICST 2024.

10. ACL 2024: "*On the Relationship Between RNN Hidden-State Vectors and Semantic Structures*" [172]. In this paper, we analyze the assumption that hidden-state vectors of RNNs form clusters. We empirically examine this assumption on different RNN architectures trained on various finite state models. We start by examining the (piecewise) linear separability of the RNN hidden-state vector with respect to automaton states. We then examine whether clustering algorithms can compute clusters that correlate with automaton states. I have contributed to every part of the paper, and

have implemented the majority of the evaluation framework. This work was accepted at Findings of ACL 2024.

1.5.3 Additional Contributions

This section contains additional publications and other academic work that was performed during the doctoral studies, but does not contribute directly to the thesis.

Additional Publications

1. DX 2021: "*Automata learning enabling model-based diagnosis*" [165]. In this paper, we outline a method that combines automata learning and fault injections. With the proposed method, learned models contain nominal and faulty behavior of the systems and can be used for model-based diagnosis. I initiated the idea behind the paper, wrote the method section of the paper and the evaluation framework, and presented it at DX 2021.
2. ICST 2021: "*Learning-based fuzzing of IoT message brokers*" [9]. In this paper, we presented a learning-based fuzzing framework. The fuzzing framework was used to fuzz-test MQTT brokers, and we found multiple bugs and security vulnerabilities. This paper is a joint work with Andrea Pferscher. I have implemented parts of the framework, and have written parts of the experimental setup section. This paper was presented at ICST 2021.
3. FMAS 2022: "*Active vs. Passive: A Comparison of Automata Learning Paradigms for Network Protocols*" [10]. In a joint work with Andrea Pferscher, we compared the effectiveness of active and passive automata learning algorithms on various communications protocols. I have implemented the required algorithms in AALPY, and have helped with the creation of the evaluation framework. This work was presented at FMAS 2022.
4. FMDT 2023: "*Mining Digital Twins of a VPN Server*" [198]. In this work, we used AALPY to infer behavioral models of IPsec IKEv1 protocol implementations. I have assisted in the setup of the automata learning framework. This work is a collaboration with Andrea Pferscher and Benjamin Wunderling and was presented at FMDT 2023.

Supervised Thesis and Projects

- Bachelor's thesis of Moritz Pistauer: In his bachelor thesis, Mortiz Pistauer designed and developed a KV-based algorithm for active inference of deterministic visibly pushdown automata. The implementation of the algorithm can be found in AALPY. I have supervised Moritz in algorithm design and development.
- Tamim Burgstaller's project on the learning behavioral models of Git version control system: For the practical part of the Model-based Testing course, Tamim developed a framework for learning behavioral models of Git version control system. I assisted Tamim through the project, namely in the design of the input and output alphabet and in the debugging. The project is open-source and available online [35], and has been accepted at the A-MOST 2024 [35].

Teaching

I assisted in the organization and lecturing in the "Quality Assurance in Software Development" course. My main responsibility was the organization of the exercises of the course. In addition, in the lecture part of the course, I held a lecture on automata learning. In the practical part of the course, students were introduced to concepts such as unit testing, mutation testing, line and branch coverage, model-based testing, property-based testing, concolic testing, and learning-based testing. I assisted in this course in the summer semesters of 2022, 2023, and 2024.

1.5.4 Thesis Structure

The thesis is structured as follows. In this chapter, we outline the motivation behind the thesis, define research questions, and present contributions. In Chapter 2. We outline the preliminaries used throughout the thesis. Firstly, we outline the basics of automata theory followed by an introduction to active and passive automata learning. Then we present the basic background on RNNs and reinforcement learning.

In Chapter 3 we describe the design decisions behind AALPY, an active automata learning library. We present its structure, interfaces, features, and use cases, and outline several optimizations that enable efficient automata learning. In Chapter 4 we describe and empirically evaluate L^*_{SMM}, the state-of-the-art active automata learning algorithm used for modeling of stochastic reactive systems.

In the following two chapters we answer RQ 2. In Chapter 5 we describe the black-box extraction method of finite-state models from RNNs. In Chapter 6 we examine the long-believed assumption that hidden-state vectors of RNN tend to form clusters that correlate with automaton states.

In Chapters 7 and 9 we answer the RQ 3. In Chapter 7, we show how automata learning can enable reinforcement learning in partially observable environments, while in Chapter 8 we show how automata learning can be used in combination with shielding to enable safe reinforcement learning. In Chapter 9 we show how automata learning can be used to construct finite-state models of potentially infinite environments. These models can then be used either to compute an explainable controller or to test Deep-RL agents.

In Chapter 10 we outline related work, focusing on advancements in automata learning and its application in the ML domain. Finally, in Chapter 11 we conclude the thesis, reflect on the research questions, and outline the future work.

2 PRELIMINARIES

In this section, we outline the basic concepts used throughout the thesis. We formalize several types of finite state machines, explain the principles behind automata learning, and provide a brief introduction to recurrent neural networks and reinforcement learning, while more specific concepts are introduced in their respective chapters.

2.1 Finite State Machines

Finite-state machines (FSM) are a concept used to represent systems with a finite number of states and transitions between those states. They can be used to model various systems, and are used in various applications, such as lexical analysis [3, 113], design of digital circuits [175, 193], network protocols [198, 234], robotics [23, 189], and much more. Automata learning algorithms encode the input-output behavior of the SUL as FSM, therefore we start by introducing the most common deterministic and stochastic modeling formalisms. We also introduce some FSM properties that provide us with a better understanding of the automata learning processes.

2.1.1 Deterministic Models

In this section, we outline deterministic regular languages used throughout the thesis, as well as some properties of regular languages that are utilized in automata learning. As a running example, we present a simplified model of a coffee machine. This coffee machine returns coffee after inserting a *coin* and pressing a *button*. If a button is pressed but no coins are added the coffee machine does not return a coffee. The number of entered coins does not influence the behavior of the coffee machine, as long as one coin is added before the press of the button.

Definition 2.1 (Deterministic Finite Automata (DFA)).
A DFA is a 5-tuple $\langle Q, I, \delta, q_0, F \rangle$*, where:*

- Q *is a finite set of states,*
- I *is a finite set of input symbols (input alphabet),*
- $\delta : Q \times I \to Q$ *is the transition function,*
- $q_0 \in Q$ *is the initial state, and*
- $F \subseteq Q$ *is the set of accepting (final) states.*

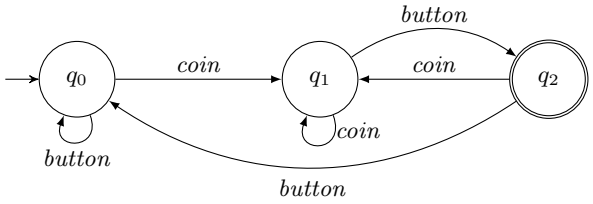


Figure 2.1: DFA representing the language of the coffee machine. This model accepts sequences that result in a coffee.

Deterministic finite automata (DFA) are a central modeling formalism in automata theory. They are a class of formal languages that can be described by regular expressions. DFAs are used to define a regular language. A word (sequence of inputs) is said to be in a language if, when executed on a DFA, it leads to an accepting state, and is rejected otherwise.

An example of a DFA can be seen in Figure 2.1. State $q2$ is accepting, and thus all sequences that end in the state $q2$ are in the "language of the coffee". For example, input sequence *(coin, coin, button)* results in an accepting state representing coffee, while *(button, coin, coin)* does not.

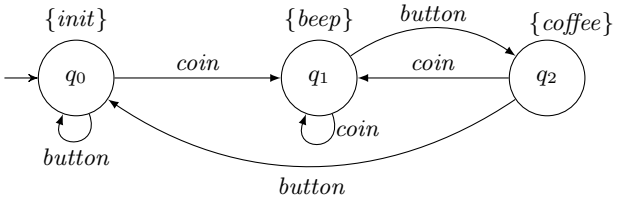


Figure 2.2: Moore machine model of a coffee machine

Definition 2.2 (Moore Machine).
A Moore machine is a 6-tuple $\langle Q, I, O, \delta, \lambda, q_0 \rangle$, where:

- *Q is a finite set of states,*
- *I is a finite set of input symbols (input alphabet),*
- *O is a finite set of output symbols (output alphabet),*
- *$\delta : Q \times I \to Q$ is the transition function,*
- *$\lambda : Q \to O$ is the output function and*
- *$q_0 \in Q$ is the initial state.*

Moore machines could be viewed as a generalization of DFAs, introducing symbolic outputs associated with each state, as opposed to the binary output (True or False) in DFAs. Moore machine with $n, n > 1$ states can have between 2 and n distinct outputs. A minimal Moore machine with one output would have a single state, and the upper bound of the output alphabet is limited by the bijective mapping of the state to the element of the output domain. Moore machines are useful when modeling more expressive systems, whose states have outputs that have some symbolic value, for example, network protocols. Note that with an empty sequence (ε) we can observe the output of the current state of Moore machines and the acceptance of the state for DFAs.

A Moore machine describing the behavior of the coffee machine can be seen in Figure 2.2. State q_0 has the output *init*, the output of state q_1 is *beep*, and in state q_2 the output *coffee* is observed. The Moore machine model of a coffee machine is more expressive than the previously shown DFA, as it also contains all sequences that can be used to get coffee, but also contains additional information such as a *beep* that confirms that a coin has been added.

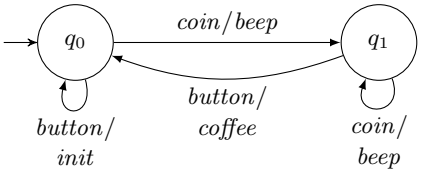


Figure 2.3: Mealy machine model of a coffee machine

Definition 2.3 (Mealy Machine).
A Mealy machine is a 6-tuple $\langle Q, I, O, \delta, \lambda, q_0 \rangle$, where:

- *Q is a finite set of states,*
- *I is a finite set of input symbols (input alphabet),*
- *O is a finite set of output symbols (output alphabet),*
- *$\delta : Q \times I \to Q$ is the transition function,*
- *$\lambda : Q \times I \to O$ is the output function and*
- *$q_0 \in Q$ is the initial state.*

Mealy machines are similar to Moore machines, in that they map the input domain to a symbolic output domain, but the mapping process differs. Whereas Moore machines associate output with a state, Mealy machines associate an output with a state-input pair. This often results in smaller models, as it might not be necessary to create a new state for every output. However, unlike Moore machines and DFAs, Mealy machines do not support an empty sequence (ε). That is, an output cannot be produced in an initial state without an explicit input.

A coffee machine modeled as a Mealy machine can be seen in Figure 2.3. The Mealy model of the coffee machine has the same input-output behavior as the Moore machine model, but with fewer states. This is the consequence of an output function that maps a state and an input to an output, thus not requiring a new state for each observed output. Mealy machines are often preferred in the context of automata learning, as their potentially smaller size could lead to more efficient learning.

Extensions of Transitions and Output Functions.

We consider input-enabled deterministic models. Given that our models are input enabled, δ and λ are total functions, that is, successor states and outputs are defined for all inputs in all states. Deterministic models define exactly one transition and one successor state for every state-input pair. This allows us to define δ and λ as mathematical functions. Output and transition functions are extended to sequences in the standard way. Let $s \in I^*$ be an input sequence and $q \in S$ a state, then $\delta(q, s) = q'$. This state is defined inductively: $\delta(q, \varepsilon) = q$ and $\delta(\delta(q, s[1]), s[> 1]) = q'$ if $|s| > 0$. For $s \in I^*$ and $q \in Q$, the output function $\lambda(q, s) = t \in O^*$ returns the outputs produced in response to s executed in state q. Formally, $\lambda(q, \varepsilon) = \varepsilon$ and $(\lambda(q, s) = \lambda(q, s[1]) \cdot \lambda(\delta(q, s[1]), s[> 1])$ if $|s| > 0$. Furthermore, let $\lambda(s) = \lambda(q_0, s)$ and $\delta(s) = \delta(q_0, s)$. Finally, we introduce traces, a combination of the state transition function and output transition function. A trace $(I, O)^+$ over deterministic automata is an alternating sequence of input-output pairs. For example, *((coin, beep), (button, coffee))* is a trace observed on the coffee machine.

Automata Theory Concepts Used in Automata Learning

Access sequences, also known as state prefixes, are sequences of inputs that can be used to reach a specific state. In automata learning prefixes are used as a form of state identification, and they are often used during conformance testing to enable model-based testing strategies. For a Moore machine shown in Figure 2.2, the access sequence of $q2$ is *(coin, button)*.

Definition 2.4 (Access Sequence).
For a state q, the set $acc(q) = \{s \in I^* \mid \delta(q_0, s) = q\}$ contains the access sequences of q. These sequences, also known as state prefixes, are the sequences leading to q, and we are often interested in the shortest access sequence leading to q.

Active automata learning uses an induced characterization set of the SUL to split states through learning. Informally, a characterization set can be seen as a (not necessarily minimal) set of sequences that when executed on all pairs of states in an automaton display a different output behavior. For example, a characterization set of the DFA shown in Figure 2.1 would be $\{\varepsilon,\ button\}$. States q_0 and q_2 could be differentiated by just observing their output with an ε, while states q_0 and q_1 cannot be differentiated just by observing their output, as they are both rejecting, but are differentiated (as they show different future behavior) with *button*.

Definition 2.5 (Characterization Set).
A finite set W is a characterization set of an automaton M iff for any pair of states q, q' in S, there is a sequence in W that can distinguish the states in the pair, or formally
$\forall q, q' \in Q : q \neq q' \rightarrow \exists w \in W : \lambda(q, w) \neq \lambda(q', w)$

When we refer to an automaton throughout the thesis, unless specified otherwise, it can be assumed that it is minimal. Active automata learning algorithms induce minimal automata by design. Informally, an automaton is minimal if no pair of states display the same future behavior, or alternatively, if its behavior cannot be encoded with an automaton with fewer states.

Definition 2.6 (Minimal Automaton).
A finite automaton M is minimal iff there exist no pair of states q, q' such that $q, q' \in Q : \forall w \in W : \lambda(q, w) = \lambda(q', w)$. That is, there does not exist a pair of states q, q' that exhibits the same output behavior for all sequences in the characterization set W.

Two automata are equivalent if they display the same behavior for all possible input sequences. As a consequence, when two minimal automata are equivalent, they will have the same state and transition structure, and can only be different in state labeling.

Definition 2.7 (Automata Equivalence).
Two minimal automata M_1 and M_2 are equivalent if there exists a bijection h between their states sets $h : Q_1 \rightarrow Q_2$ such that the following conditions hold:

- *$h(q_{0_1}) = q_{0_2}$, i.e., the initial states are mapped to each other.*
- *For all $q \in Q_1$ and $i \in I$, $h(\delta_1(q, i)) = \delta_2(h(q), i)$, i.e., the transition functions are preserved under the mapping h.*
- *Additional property for the equivalence of DFAs and Moore machines*
  - *For all $q \in S_1$, $h(q) \in S_2$, the output mapping/state acceptance is preserved*

Automaton equivalence between automata M_1 and M_2 is easy to determine if both models are known. However, if one model is known, and the other is a black box (like in active automata learning), determining automaton equivalence is undecidable. However, we introduce the concept of black-box observational equivalence.

Definition 2.8 (Black-Box Observation Equivalence).
Two automata M_1 and M_2 with the same input and output alphabets are observation equivalent if $\forall s \in I^* \mid \lambda_{M_1}(q_0, s) = \lambda_{M_2}(q_0, s)$, that is if they display same output behavior for all input sequences.

In practice, we take the testing approach to observational equivalence. We say that two automata are observation equivalent if they display the same behavior for all sequences of the test suite TS. For example, the Moore and Mealy machine of coffee machines are observationally equivalent.

An input sequence that displays an observational difference between two models is called a counterexample. Counterexamples are used in automata learning to refine the learned model, and once a counterexample between the induced model and black-box system cannot be found, the learning stops.

Definition 2.9 (Counterexample to Observation Equivalence).
Given two automata M_1 and M_2, an input sequence $cex \in I^*$ is said to be a counterexample to observation equivalence if $\lambda_{M_1}(cex) \neq \lambda_{M_2}(cex)$.

Finally, the concepts of sequence prefixes and suffixes are used extensively in automata learning. Given an input sequence (i_1, i_2, i_3), its prefixes are $\{(i_1), (i_1, i_2), (i_1, i_2, i_3)\}$, while its suffixes are $\{(i_3), (i_2, i_3), (i_1, i_2, i_3)\}$. Note that some definitions of prefixes and suffixes also consider an empty sequence (ε).

Definition 2.10 (Prefixes of a Sequence).
For an input sequence $s = s[1]s[2]\dots s[n], s[i] \in I$, the set of prefixes of s, is denoted by pref(s) and defined as follows: pref(s) = $s[1], s[1]s[2], \dots, s[1]s[2]\dots s[n]$, where s[i] represents i^{th} element of an input sequence.

Definition 2.11 (Suffixes of a Sequence).
For an input sequence $s = s[1]s[2]\dots s[n], s[i] \in I$, the set of suffixes of s, is denoted by suff(s) and defined as follows: suff(s) = $s[n], s[n-1]s[n], \dots, s[2]\dots s[n], s[1]s[2]\dots s[n]$.

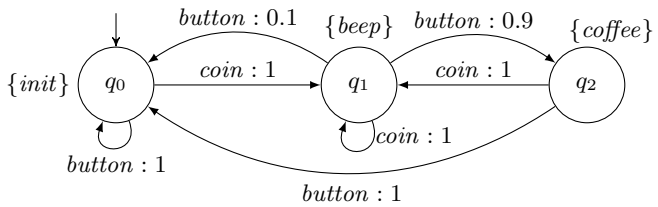


Figure 2.4: An MDP of the faulty coffee machine

2.1.2 Markov Decision Processes

MDPs are used to model probabilistic reactive systems [19]. An MDP reacts to the inputs stochastically, that is, an input from a given state might result in more than one output. Compared to purely non-deterministic automata, outputs are sampled from probability distributions assigned to a state-input pair. Formally, MDPs are defined as follows:

Definition 2.12 (Markov Decision Process (MDP)).
A Markov decision process is a 6-tuple $\langle Q, I, O, \delta, \lambda, q_0 \rangle$, where

- *Q is a finite set of states,*
- *I is a finite set of input symbols (input alphabet),*
- *O is a finite set of output symbols (output alphabet),*
- *$\delta : Q \times I \to$ Dist(Q) is the probabilistic transition function,*
  - *where Dist(Q) is a set of all probability distributions over Q*
- *$\lambda : Q \to O$ is the output function and*
- *$q_0 \in Q$ is the initial state.*

Like for deterministic automata, MDPs are input-enabled and λ and δ are defined for all $q \in Q$ and $i \in I$. In the context of MDPs, λ is also known as a labeling function, as it assigns labels (outputs) to states. Unless stated otherwise, we consider deterministically-labeled MPDs. Deterministically-labeled MDPs assign at most one successor to each state and input-output pair, or formally:

$$\forall q, q', q'' \in Q, \forall i : (\delta(q, i)(q') > 0 \wedge \delta(q, i)(q'') > 0) \to (q' = q'' \vee \lambda(q') \neq \lambda(q''))$$

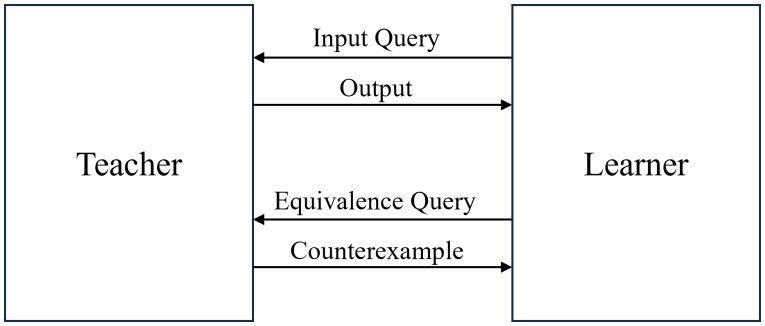


Figure 2.5: Minimally adequate teacher (MAT) framework

We extend the transition and output functions of MDPs analogous to the ones used in deterministic models. The difference to the deterministic formalism is that we consider MDP traces of the form $O \cdot (I, O)^+$, that is, input-output sequences are prepended with the output of the initial state of the model. Given that MDPs are deterministically labeled, a trace $O \cdot (I, O)^+$ uniquely identifies a reached state, as shown in the next paragraph.

An example of an MDP encoding the behavior of a faulty coffee machine can be seen in Figure 2.4. This coffee machine behaves similarly to the one described in the previous section, but once the *coin* has been entered, a pressing of a *button* might result in a coffee (with a 90% probability) or the coffee machine experiences a fault and is reset to an initial state. This MDP is deterministically labeled, as from state $q1$ a *button* action can lead to two transitions, which are uniquely identified by their input-output values (*button/init* and *button/coffee*). Once the output is determined (by sampling according to transition probabilities), a unique input-output pair determines the next state, e.g., *button/coffee* leads to $q2$.

Finnaly, when MDPs are used to solve some task we need to compute a strategy that selects appropriate actions given the current state of the MDP. Such a strategy is called a policy. A policy resolves the non-deterministic choice of actions in an MDP. It is a function mapping trajectories to distributions over actions. We consider *memoryless* policies that take into account only the last state of a trajectory, i.e., policies $\pi : Q \to Dist(I)$. A *deterministic* policy π always selects a single action, i.e., $\pi : \mathcal{S} \to I$.

2.2 Active Automata Learning

In her seminal work on active automata learning [14], Angluin introduced an active automata learning framework, which she formalized with the Minimally Adequate Teacher (MAT) framework. Angluin also proposed the L^* algorithm, which was designed to infer unknown regular languages and encode them as DFA. Since then, the MAT framework has served as a basis for many active automata learning algorithms, and L^* has been improved and extended to many different formalisms, such as Mealy [66] and Moore machines and MDPs [237].

To introduce active automata learning, we first describe the minimally adequate teacher framework, outline the role of conformance testing in active automata learning, and finally describe the inner workings of L^* on the adapted coffee machine example.

2.2.1 Minimally Adequate Teacher Framework

The minimally adequate teacher framework, as depicted in Figure 2.5, describes the interactions between the Learner and the Teacher. In the MAT framework, a learner (automata learning algorithm) iteratively builds a model of the SUL by asking the teacher two kinds of queries: input queries and equivalence queries. The learning process starts with the learner asking input queries.

Input queries (also known as membership queries) are a sequence of discrete inputs that are executed on the SUL, and a corresponding discrete output sequence is obtained. In the majority of active learning

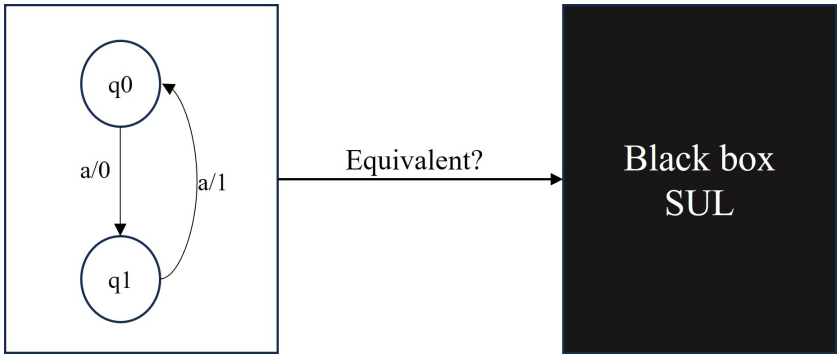


Figure 2.6: Conformance testing against black-box SUL

algorithms, all input queries are executed from an initial state. For example, after executing the input query *(coin, button)*, the outputs *(beep, coffee)* are observed. Input queries are implemented with the following operations: *reset* and *step*. Since each input query is executed from the initial state of the SUL, the SUL needs to be *reset* before a query execution. Implementation of the reset operation is domain-specific. For example, when learning network protocols, one might need to completely reboot the device to bring it to the consistent initial state. And since an input query is just a sequence of steps on the SUL, a *step* operation executes a single step on the SUL, that is it executes an action defined in the input alphabet on the SUL. When sufficient information about the SUL is obtained through input queries, the learner constructs a model of the SUL, called a hypothesis. The hypothesis is then passed to an equivalence query, which checks if the hypothesis exactly represents the SUL's behavior.

The equivalence query is answered by an equivalence oracle. The equivalence oracle either returns a counterexample, which is then used to extend the hypothesis by asking additional input queries, or it returns no counterexample, at which point the learning stops. In practice, the equivalence query is answered with conformance testing: a test suite consisting of input queries is executed both on the SUL and on the hypothesis. If an example of non-conformance is found, i.e. SUL and hypothesis display different output behavior for the same input query, it is used as a counterexample.

2.2.2 Conformance Testing

Consider the following problem depicted in Figure 2.6. Assume we have modeled the system with a hypothesis as shown on the left-hand side of the figure, and we want to prove its equivalence to a black-box system with which we can interact. Given that we usually assume no knowledge about the structure of the black box, this question remains undecidable. Even if we were to test all input sequences of length n, an input sequence of length $n + 1$ might reveal the difference in behavior between the hypothesis and SUL. However, if the maximum possible number of states of the SUL is known, the conformance-checking problem becomes decidable.

To provide a practical answer to the conformance checking we turn to conformance testing. The role of conformance testing is to test whether an implementation conforms to the specification. In the context of automata learning, conformance testing is used to answer an equivalence query as in practice the existence of the perfect teacher (one that returns a shortest counterexample between SUL and the hypothesis) cannot be assumed.

Conformance testing strategies attempt to find a finite test-suite *TS* that can determine whether the black-box SUL and the current hypothesis are equivalent. Given that equivalence cannot be proven, we view the equivalence problem from a testing perspective. A test suite *TS* establishes the equivalence between the hypothesis and SUL if all test cases are passing. We define a test case as *passing* if the behavior of the hypothesis conforms to the behavior of the SUL. Alternatively, if the test case shows different behavior on the SUL and on the black box, we dub it as *failing*. If all test cases in *TS* are *passing*, we say that the hypothesis conforms to the SUL.

For a finite test suite *TC*, hypothesis H and a black-box *SUL*, we define an implementation relation [247] *imp* as:

$$H \textbf{ imp } SUL \Leftrightarrow \forall\, tc \in TS :\ H \textbf{ passes } tc$$

The implementation relation is satisfied if all test cases in *TS* are passing. It is important to note that the implementation relation only assesses the equivalence between H and *SUL* based on the *TS*. This brings us to a fundamental question in automata learning and conformance testing: "How do we design a model-based testing strategy that is able to generate a finite *TS* that gives us confidence in this notion of equivalence?"

In practice, there are several ways of designing an equivalence oracle that generates a test suite, but we chose to differentiate them in the following categories:

- **W-Method** [49]: This equivalence oracle provides formal guarantees that *TS* covers all possible behavior of the SUL, thus it can be used to formally prove the equivalence between H and SUL. However, this oracle requires that the user knows the maximum number of states of the SUL. In practice, the usage of the W-Method is often not feasible as the *TS* has an exponential number of test cases w.r.t. the number of states of the SUL.
- **Purely Random Oracles**: Purely random oracles create test cases by uniformly sampling inputs from the input alphabet. They are often used in practice as the size of *TS* can be defined a priori, and they can even be extended to provide probably approximately correct (PAC) guarantees [252]. However, purely random oracles might not sufficiently explore "hard-to-reach" regions of the SUL.
- **Model-guided Oracles**: These oracles use the inferred structure of the current hypothesis during the test-case generation. They often feature test-case generation strategies that maximize the state or transition coverage of the current hypothesis. Furthermore, they could include elements of the characterization set to exploit domain knowledge in the search for the counterexample.

Algorithm 2.1 Test case generation with W-Method

Input: Hypothesis *hyp* = $\langle Q, q_0, I, O, \delta, \lambda \rangle$, maximum number of states m
Output: A test suite *TS*

1: *stateCover* $\leftarrow \{seq \mid q \in Q, seq \in acc(q)\}$
2: *transitionCover* $\leftarrow$ *stateCover* $\cdot\, I$
3: *middle* $\leftarrow \bigcup_{d=0}^{m-|Q|} I^d$
4: $W \leftarrow$ buildCharacterising(*hyp*)
5: $TS \leftarrow$ *transitionCover* $\cdot$ *middle* $\cdot\, W$
6: **return** TS

W-Method. The W-method was initially developed by Chow [49] as a method for a test-case generation from possibly incorrect program designs. It was used to test such programs with respect to a specification. The described use case is fully transferable to our context: our hypothesis is possibly incorrect and we want to find cases of non-conformance with respect to the SUL. The W-method requires that the maximum possible number of states of the SUL (m) is defined a priori. With this assumption, the W-method generates an exhaustive test suite that formally proves the equivalence between the hypothesis and the SUL.

Algorithm 2.1 describes the test suite generation with the W-method. All test cases generated by the W-method consist of three parts: a prefix reaching a transition of a hypothesis, a potentially empty middle part, and a characterizing sequence. Remember that the characterizing set W is able to distinguish any pair of states of the hypothesis *hyp*. Executions of W from all transitions of the hypothesis can detect

Algorithm 2.2 Test case generation with Random Word equivalence oracle

Input: Input alphabet I, number of test cases n, minimum tc_{min} and maximum tc_{max} test case length
Output: A test suite *TS*

1: $TS \leftarrow \{\ \}$
2: **while** $|TS| < n$ **do**
3: $tcLength \leftarrow$ uniform(tc_{min}, tc_{max}) ▷ Test case length is sampled from uniform distribution
4: $randomTestCase \leftarrow$ randomSequance(I, *tcLength*) ▷ Uniformly sample *tcLength* inputs from I
5: $TS \leftarrow TS \cup \{\ randomTestCase\ \}$
6: **return** *TS*

all cases of non-conformance between *hyp* and SUL with at most $|hyp|$ number of states [49]. The introduction of the *middle* part is necessary to detect additional states. In practice, the usage of the W-method is not feasible due to the exponential size w.r.t to m

In addition, in the context of automata learning we can rarely assume the maximum number of states of the learned model. This assumption is problematic for the following reason: suppose we want to learn a model of some black-box system that is *supposed* to have ten states (according to some specifications). However, the ten states assumption is made w.r.t. the specification, not w.r.t to the implementation. That is, the implementation should implement a specification that has 10 states, but implementations often contain unexpected bugs that increase the number of states of the learned model, as observed e.g. in the active learning of MQTT protocols [234].

Purely Random Oracles. Purely random equivalence oracles are often used to answer the equivalence query. They do not offer formal guarantees but are often sufficient to display non-conformance between the hypothesis and the SUL. This type of equivalence oracle does not consider the structure of the hypothesis during the test case generations, but uniformly samples inputs from the input alphabet.

The most notable examples of such oracles are random walk and random word equivalence oracles. Both oracles are implemented in LearnLib [111] and AALPY [166], and we consider their implementation as a specification of these oracles. In both oracles a maximum number of test cases is predefined, and all test cases are sequences of uniformly sampled elements of the input alphabet. Those two oracles differ in how they define the length of individual test cases: in a random walk oracle, the test case length is controlled through the stopping probability, while in the random word oracle, the test case length is uniformly sampled from the range defined by tc_{min} and tc_{max}. Algorithm 2.2 shows the test-case generation with a random word equivalence oracle. Note that both methods yield similar results, as long as the maximum test case length is large enough w.r.t. the length of the potential counterexample. Tappler [233] closely examined the capabilities of random testing in his thesis, and concluded that random testing is often a sufficient conformance testing strategy, but since it does not exploit the knowledge about the hypothesis it tends to not find all counterexamples for bigger or more complex models.

Algorithm 2.3 Test case generation with Random W-Method equivalence oracle

Input: Hypothesis $hyp = \langle Q, q_0, I, O, \delta, \lambda \rangle$, walks per state n, random walk length r
Output: A test suite *TS*

1: $TS \leftarrow \{\ \}$
2: $prefixes \leftarrow \{seq \mid q \in Q, seq \in acc(q)\}$
3: $W \leftarrow$ buildCharacterising(*hyp*)
4: **for** p *in prefixes* **do**
5: **for** $i \leftarrow 1$ **to** n **do do**
6: $tc \leftarrow p \cdot$ randomWalk(hyp, r) $\cdot$ randomChoice(W)
7: $TS \leftarrow TS \cup \{\ tc\ \}$
8: **return** TS

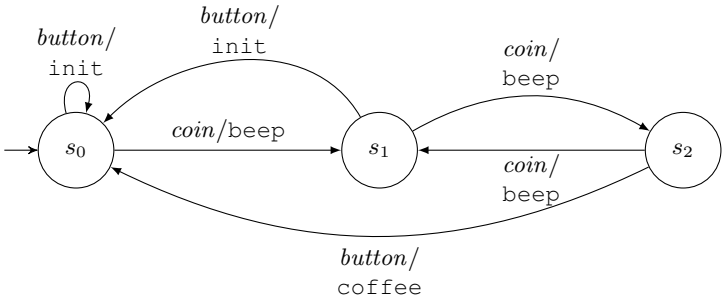


Figure 2.7: Model of a coffee machine that returns coffee after receiving an even number of coins

Model-guided Oracles. This final class of equivalence oracles builds upon the W-method and purely random testing. The majority of such oracles implement some notion of state or transition coverage. The most notable example is the Random-W method [111, 167]. We present the implementation of the Random-W method as implemented in AALPY. Algorithm 2.3 depicts the algorithm used to create a test suite with the Random-W method. Random-W method provides a state coverage criterion, as each state of the hypothesis serves as an origin for n test cases. Each test case therefore consists of a prefix to a state, a random middle part (random word), and a uniformly chosen element of the characterization set (Line 6). In contrast to the W-method, Random-W scales well but does not provide the same formal guarantees. However, in practice, the usage of Random-W and similar oracles provides us with the highest confidence that (almost) all counterexamples between the hypothesis and SUL are found. Other examples of equivalence oracles that provide coverage guarantees are outlined in Chapter. 3.

We further reason about further conformance testing strategies and elaborate on practical and theoretical aspects of conformance checking in automata learning throughout the thesis. More concretely, in Chapter 3 we outline several conformance testing strategies implemented in AALPY, and in Chapter 5 we reason which conformance testing strategy is the best for the extraction of models from RNNs.

2.2.3 Learning Mealy Machines with L^*

We present a variation of the original L^* algorithm that captures the input-output behavior of the SUL and encodes it as a Mealy machine [220]. Furthermore, we extend the L^* algorithm with the Rivest-Schapire (RS) [215] counterexample processing. L^* with RS counterexample processing is more efficient (with respect to the number of interactions with the SUL) than the original L^* and it became a de-facto standard L^* implementation. We describe the inner workings of L^* and use the adapted model of a coffee machine shown in Figure 2.7 as a running example. The developer of this coffee machine had an irrational fear of odd numbers, so this coffee machine only produces coffee if an even number of coins is entered. This three-state Mealy machine is introduced as L^* would identify the correct structure of other previously introduced models in just one learning round.

Abstractly, the goal of L^* can be seen as the identification of states (and transitions between those states) of the SUL. To identify states of the SUL, a criterion that differentiates states needs to be defined: a characterization set. As stated in Sect. 2.1.1, a characterization set is a set of input sequences that is able to differentiate between all states of the automaton. Therefore, active automata learning is concerned with the identification of the characterization set, as it can be used to identify the states of the model.

To enable the identification of states with the characterization set, L^* maintains a data structure called an observation table. An observation table is a table-based structure that is used to construct the hypothesis of the SUL. It contains all outputs observed during learning, and it is used to identify states and transitions during learning. Formally we define the observation table as:

Definition 2.13 (Observation Table).
An observation table τ for learning of Mealy machines $\langle Q, q_0, I, O, \delta, \lambda\rangle$ is a triplet $\langle \Gamma, E, T\rangle$, where

- $\Gamma \subseteq I^*$ *is a prefix closed set of access sequences*
- $E \subseteq I^+$ *is a suffix closed set of characterizing sequences, and*
- $T : \Gamma \times E \to O^+$ *is an output mapping function.*

Table 2.1: Observation table for a coffee machine model from Figure 2.7.

		E		
	Prefixes	coin	button	coin, button
S	ε	beep	init	beep, init
	coin	beep	init	beep, coffee
	coin, coin	beep	coffee	beep, init
$S \cdot I$	button	beep	init	beep, init
	coin, button	beep	init	beep, init
	coin, coin, coin	beep	init	beep, coffee
	coin, coin, button	beep	init	beep, init

Table 2.1 depicts an observation table that is used to construct a Mealy machine representation of the coffee machine (Figure 2.7). The observation table can be split into two main parts: rows Γ that identify states and transitions, and characterization set E that is used to differentiate states. During learning, the observation table is populated by asking input queries of the form $s \cdot e,\ s \in \Gamma,\ e \in E$, and the output of $s \cdot e$ is saved in the table cell with the output mapping $T(s,\ e)$. Note that cells in the observation tables (Tab. 2.1 and Tab. 2.2) contain all outputs observed when executing an input sequence e from a state reached with an input sequence s.

The rows of the observation table are further split into two categories: S and $S \cdot I$ sets. The S set is used to represent states of the hypothesis, and the $S \cdot I$ (also known as extended S) set identifies transitions. The extended S is defined as follows: $s \cdot i \mid i \in I,\ s \in S$, that is, we extend each prefix of S with an element of an input alphabet and add it as a row in the set $S \cdot I$. Consider the observation table shown in Tab. 2.1. The S set contains three elements, that is prefixes of three states, that are distinguished by their response to the elements of the E set, that is by their future behavior. A row in the observation table can be interpreted as a collection of state continuations (futures) that uniquely determines a state.

Transitions between states are identified as follows: given a state whose a prefix found in S, to identify its transition for an $i \in I$, we identify a reached state by asking continuation queries with the elements of the E set. More concretely, consider the state identified by the prefix $(coin)$ in Table 2.1. To identify the transition for the input $button$ from that state, we identify the row denoted by *(coin, button)* in the observation table, and match the value of said row to a row in the S set. Function *row(x)* returns all values of the E set for the row reached with the prefix x. Given that row(*(coin, button)*) evaluates to *((beep), (init), (beep, init))*, we determine that the transition from the state reached by the prefix*coin* for the input *button* leads to the state reached by the prefix ε, as its row values are also *((beep), (init), (beep, init))*. The output for this transition is defined by observing the value of the action $button$ in the row *(coin)*, which we denote in following listing with $row(s)[i]$, where s is the state prefix and i the input element of an input alphabet.

Therefore we can encode a model from an observation table as follows:

- $Q = row(s) : s \in S$
- $q_0 = row(\varepsilon)$
- $\delta(row(s), i) = row(s \cdot i)$
- $\lambda(row(s), i) = row(s)[i]$.

Finally, the observation table has to satisfy a *closedness* and *consistency* properties that enable model creation. The observation table is said to be closed, if for each row in an extended S set exists a matching

Algorithm 2.4 Abstract view of L^* algorithm

Input: SUL Interface capable of answering input queries (implements *step* and *reset*)
Output: Final hypothesis *hyp*

```
1: obs ← initializeObservationTable()
2: while True do
3:     obs ← closeObservationTable(obs)
4:     hyp ← constructHypothesis(obs)
5:     cex ← equivalenceQuery(hyp)
6:     if cex is None then
7:         return hyp
8:     suffixes ← processCounterexample(cex)
9:     obs ← extendObservationTable(suffixes)
```

row (row with the same responses to the E set) in the S set. This property ensures that all transitions have a destination state, as a destination of the transition is identified with a matching row in the S set. If the closedness property is violated, that is a row in $S \cdot I$ set does not have a match in the S set, we move that row to the S set and then extend the $S \cdot I$ to account for its transitions. The observation table is consistent if any two rows in the S match, that is $row(s_1) = row(s_2)$, their continuations also need to be the match. However, we do not elaborate further on this property at this moment, as observation table consistency is implicitly maintained in L^* with advanced counterexample processing techniques.

The Learning Algorithm. With a defined data structure used by L^*, let us turn our attention to the L^* algorithm itself. The abstract view of the algorithm is shown in the Algorithm 2.4. As previously mentioned, the learning is done by interacting with the system, and therefore the algorithm requires an implementation of the interface that can perform input queries on the SUL.

The algorithm starts by initializing the empty observation table. The observation table is initialized with a single element in the S set (ε), and with elements of the input alphabet in E. With an initialized observation table we enter a loop that is exited only when the equivalence oracle deems that the hypothesis conforms to the SUL, that is, it cannot find any counterexamples. A single iteration of the loop is known as a learning round, as it contains a round of learning, followed by a conformance checking that either indicates that further learning is required (thus entering a new learning round), or that the hypothesis conforms to the SUL.

In this loop, the observation table is populated by asking input queries until it is closed. The closed observation table is used to construct a hypothesis, which is then passed to the equivalence oracle. If the counterexample is found, it is processed with the *RS* counterexample processing. Such suffix-based counterexample processing strategies split the counterexample into a prefix and a suffix. The identified suffix is then used to extend the observation table by adding it to the E set. Once an observation table is extended, all new columns in the observation table are populated by asking input queries, which will violate the closedness property. When a row in the $S \cdot I$ set does not have a matching row in the S set, we move it to the S set and further extend the $S \cdot I$ with its continuations. Then we close the extended observation table and repeat the process until no more counterexamples can be found.

Counterexample Processing. Finally, let us briefly discuss counterexample processing. Counterexample processing strategies optimize automata learning by minimizing the number of input queries asked during the learning process. Original L^* algorithms featured prefix-based counterexample processing that added all prefixes of the counterexample to the S set. However, Rivest and Schapire [215] were the first to develop suffix-based counterexample processing. RS and other suffix-based counterexample processing [107, 220] strategies split the counterexample into a prefix v and a distinguishing suffix w. The found suffix can be used to distinguish at least two states of the hypothesis, and given that it was a

Algorithm 2.5 Rivest-Schapire counterexample processing

Input: Hypothesis *hyp*, SUL interface, counterexample *cex*
Output: Distinguishing suffix w

```
1:  lower ← 2
2:  upper ← len(cex) - 1
3:  while True do
4:      mid ← ⌊(lower + upper)/2⌋
5:      v ← cex[:mid]
6:      w ← cex[mid:]
7:      executeSequence(hyp, v)          ▷ Execute v on the hypothesis to change current state of the hyp
8:      shortPrefix = hyp.currentState.prefix          ▷ Get prefix of reached state
9:      mq = shortPrefix · w
10:     out_hyp = executeSequence(hyp, mq)
11:     out_SUL = executeSequence(sul, mq)
12:     if out_hyp eq out_SUL then
13:         lower ← mid + 1
14:         if upper < lower then
15:             return cex[mid + 1 :]
16:     else
17:         upper ← mid - 1
18:         if upper < lower then
19:             return cex[mid :]
```

counterexample to the conformance between SUL and the hypothesis, once it is added to the observation table it will lead to the increase of S.

Rivest-Schapire counterexample processing algorithm is shown in Algorithm 2.5. In RS counterexample processing, the search for the splitting point between a prefix and distinguishing suffix is done with binary search over the counterexample. The fundamental idea behind RS is the following: a counterexample *cex* is split in a prefix v and a suffix w (lines 5 and 6). Prefix v is then substituted with *shortPrefix*, which is a prefix of a state reached when executing v on the hypothesis (lines 7 and 8). A new query of the form *shortPrefix* $\cdot$ w is then executed on both the hypothesis and on the SUL (lines 10 and 11), and based on the received outputs, the splitting point used to split v and w is moved [228]. RS introduces a small overhead to counterexample processing as it queries the SUL. However, this overhead is negligible compared to the overall reduction in the total number of learning queries that RS brings to L^* [8].

Complexity and Runtime. L^* with *RS* counterexample processing, as well as TTT [109] are the most efficient active automata learning algorithms [8]. They both require O(kn^2 + n log m) input queries in the worst case, where k is the size of the input alphabet, n is the final size of the learned model, and m is the length of the longest counterexample. However, this complexity measure considers only the input queries asked during the learning process, not during the equivalence checking. Depending on the conformance testing strategy used during learning, the majority of the interaction with the SUL might be in the equivalence checking. Berg et al. [24] consider conformance testing "the true bottleneck of automata learning". However, based on our experience we slightly extend their observation and claim that the true bottleneck of active automata learning is the interaction time with the SUL. Total interaction overhead is distributed across learning and conformance testing in proportion to the total number of input queries asked in each stage. This ties in with the observation made by Berg, since usually the majority of input queries are asked in the equivalence check, therefore making the conformance testing the most resource-intensive part of the automata learning process. The impact of the interaction time with the SUL can better be understood with the following example: when learning behavioral models of

Table 2.2: Initial observation table for the coffee machine shown in Figure 2.7.

		E	
	Prefixes	coin	button
S	ε	beep	init
$S \cdot I$	coin	beep	init
	button	beep	init

an MQTT protocol [9], the runtime required to learn the three-state MQTT model was ∼260 seconds. For comparison, with AALPY, we can learn a 2000 state simulated Mealy machine in ∼1 second.

Example Run of L^* on the Coffee Machine from Figure 2.7. L^* starts with the creation of an empty observation table, with ε in the S set and input alphabet (*coin, button*) in the E set[1]. Continuations of ε are added to the $S \cdot I$ set, that is rows with prefixes *coin* and *button*. Note that we usually omit the ε prefix from the continuations of ε. Cells of the observations table are then populated with input queries of the form $s \cdot e, s \in S \cup S \cdot I, e \in E$. Since the resulting observation table (shown in Tab. 2.2) is closed, we create an initial hypothesis that is passed to the equivalence oracle. The equivalence oracle returns a counterexample *('coin', 'coin', 'coin', 'coin', 'button')*, which is then split into a prefix *(coin, coin, coin)* and a suffix *coin, button* with *RS* counterexample processing. The found distinguishing suffix is then added to the E set. After the addition of a found suffix, the row in identified with the prefix $(coin)$ displays a closedness violation and is then moved to the S set. Continuations of that row are then added to the $S \cdot I$, and when populated, reveal another closeness violation: *(coin, coin)*. That row is then also moved in the S set. After the extension of that row in the $S \cdot I$ set, the observation table is closed and matches the one found in Tab. 2.1. The second hypothesis is then passed to an equivalence oracle, that is unable to find any counterexample, and the learning algorithm returns the second hypothesis as the learned model.

2.3 Passive Automata Learning

Passive automata learning algorithms infer models of the SUL not by interaction, but from provided system traces. The learned models are dependent on the provided data, unlike in active automata learning where the SUL can be queried for more information. However, this apparent downside has not hindered the applicability of passive automata learning in various domains [10, 56, 64].

In the remainder of this section, we outline the working principles behind deterministic (RPNI) and stochastic (IOALERGIA) passive learning algorithms. Both algorithms share the same working principle: state-merging over the input-output prefix tree (IOPT). An example of IOPT can be seen at the left-hand side of Figure 2.8. Both algorithms can be formalized with the red-blue framework [58]. In this framework, the passive learning algorithm merges nodes of the IOPT while keeping track of two types of nodes: red nodes and blue nodes. Red nodes are processed nodes that are identified as distinct states of an automaton, while blue nodes are nodes of the IOPT that are next in the processing queue.

Algorithm 2.6 shows the main building blocks of the red-blue passive automata learning framework. The algorithm puts the root node of the IOPT into the red set and its successors in the blue set. The algorithm halts when all nodes of the IOPT are processed, that is when the blue set is empty. In the red-blue framework, blue nodes are checked for compatibility with red nodes. If a blue node and a red node are compatible, they are merged, and if a blue node is not compatible with any red nodes it is added to the red set. That is, if the current states of the automaton (red nodes) cannot be matched with a blue node, it is implied that a new state is identified. At the end of the algorithm, the set of red nodes is

[1]Were we to learn a DFA or a Moore machine, we would initialize E with $\{\varepsilon\}$, and during hypothesis construction that column would be used to determine state acceptance/output.

Algorithm 2.6 Red-blue framework for passive automata learning

Input: Input-output prefix tree *IOPT*
Output: Learned model M

```
1:  red ← {IOPT.rootNode}
2:  blue ← IOPT.rootNode.successors
3:  while blue ≠ { } do
4:      lexMinBlue ← getLexMin(blue)          ▷ Get lexicographically minimal node from blue set
5:      blue ← blue \ {lexMinBlue}            ▷ Remove lexMinBlue from blue set
6:      merged ← False
7:      for r ∈ red do
8:          if compatible(r, lexMinBlue) then
9:              merge(r, lexMinBlue)
10:             merged ← True
11:             break
12:     if not merged then
13:         red ← red ∪ { lexMinBlue }
14:         blue ← blue ∪ {b ∈ lexMinBlue.successors | b ∉ red}
15: M ← toAutomaton(red)
16: return M
```

transformed into an automaton. Note that due to its reliance on the provided data set, the resulting model might not be input-complete, meaning that some states of the automaton have undefined transitions.

The red-blue framework consists of three abstract functions, that are implemented differently by deterministic and stochastic passive automata learning algorithms. These abstract functions are:

- **compatible(*red*, *blue*)**: This function evaluates whether *red* and *blue* nodes are deemed equivalent by some distinguishing criterion.
- **merge(*red*, *blue*)**: This function merges compatible *red* and *blue* nodes and their respective subtrees/continuations in the IOPT.
- **toAutomaton(*red*)**: This function encodes the set of *red* nodes as a finite-state automaton.

2.3.1 Deterministic Passive Automata Learning

Regular positive-negative inference (RPNI) [58, 187] was developed as a method of inference of DFAs from a finite set of positive and negative sequences (traces). Positive sequences are sequences accepted by the language, while the language rejects negative sequences. Since its development, RPNI has been extended to other deterministic formalisms, most notably for Mealy and Moore machines [81].

Like other algorithms from the red-blue framework, RPNI starts by constructing an input-output prefix tree from provided sequences. An example of such an IOPT can be seen on the left-hand side of Figure 2.8. In the following listing, we provide details about RPNI's implementation of abstract functions that enable passive learning of deterministic systems:

- **compatible(*red*, *blue*)**: *red* and *blue* nodes are compatible if they display the same future behavior for all common input-output sequences. In the RPNI implementations, this check is performed by first merging the *red* and *blue* nodes and checking if the resulting model still accepts all sequences in the provided data set. If non-deterministic behavior is observed during the merge, or if the merge results in a model that is not consistent with the provided data set, *red* and *blue* nodes are not compatible and the merge is reverted.
- **merge(*red*, *blue*)**: A merge is performed by recursively folding all descendants of the blue node into the red tree.
- **toAutomaton(*red*)**: Encoding the *red* node set to deterministic automata is straightforward. The root node serves as an initial state of the automaton, and transitions follow the transitions between

red nodes. Notice that the resulting IOPT found on the right-hand side of Figure 2.8 is equivalent to the Mealy machine found in Figure 2.3.

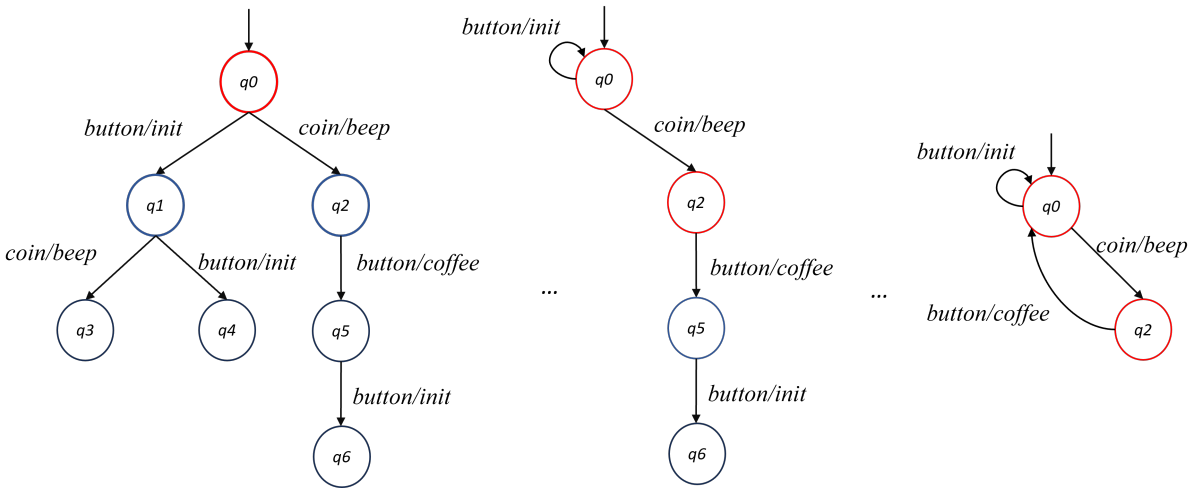


Figure 2.8: Stages of the execution of the RPNI algorithm

RPNI example on the Mealy machine model of the coffee machine. Figure 2.8 shows the progress of the RPNI algorithm. It starts with the building of an IOPT from the provided sequences. Given that we display a Mealy machine variation of RPNI, nodes are not labeled, but transitions are labeled with input/output pairs. Such an IOPT can be seen on the left-hand side of the figure. Note that the initial node is red, and its successors (children) are blue. The algorithm first checks whether nodes q_1 (left successor of q_0) and q_0 are compatible. Since states q_0 and q_1 display the same future behavior for all common continuations we deem them as compatible and proceed with the merge. Since q_1 is folded into q_0, a transition that was previously leading to q_1 is now a self-loop. In the next step, RPNI checks if q_0 is compatible with q_2. They are not compatible, since input *button* evaluates to *init* in q_0, but to *coffee* in q_2. Since they are not compatible, q_2 is added to the red set. The middle figure in Figure 2.8 depicts the current state of the IOPT. This process is repeated for the remaining nodes, and the algorithm returns a model shown at the right-hand side of Figure 2.8. Note that the resulting model is not input complete, that is, a transition for an input *coin* is not defined in q_2. This behavior is not uncommon in passive automata learning if the provided data set does not cover all the behavior of the SUL.

2.3.2 Stochastic Passive Automata Learning

ALERGIA [41] is a passive automaton learning algorithm that infers stochastic grammars via state merging. More precisely, it was developed to infer stochastic finite automata from discrete sequential data. IOALERGIA [144] is an adaptation of the original algorithm that is able to model stochastic input-output data as MDPs. In this thesis, we put more emphasis on IOALERGIA, since we are concerned with modeling of an SUL's input-output behavior.

IOALERGIA shares similarities with RPNI in that it also adheres to the red-blue framework. However, the presence of stochastic behavior significantly changes the three previously mentioned abstract functions of the red-blue framework.

The first notable difference is in the creation of the IOPT. More precisely, IOALERGIA uses an input-output frequency prefix tree (IOFPT). An example of such a tree can be seen on the left-hand side of Figure 2.9. In IOFPTs, nodes are labeled with outputs (as we are learning an MDP that has outputs tied to a state) and transitions are labeled with inputs and the frequency with which the transition was observed in the provided data set. Frequencies are used for two purposes: to help differentiate states during the learning process and to provide data used to compute transition probabilities in the learned model. With

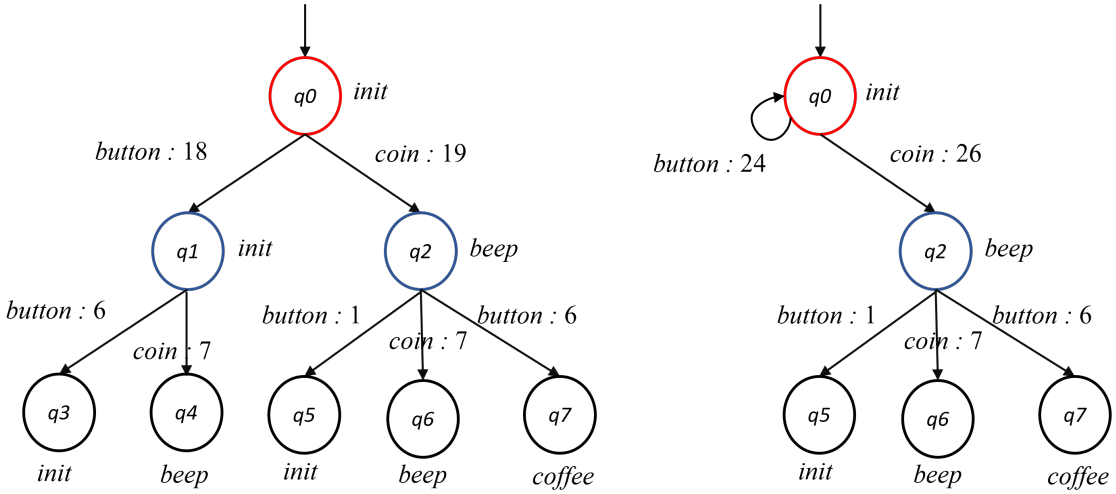


Figure 2.9: Initial IOFPT (left) and IOFPT after a first merge (right)

the constructed IOFPT, IOALERGIA follows the red-blue framework described in Algorithm 2.6, and implements its abstract functions as follows:

- **compatible(*red*, *blue*)**: *red* and *blue* nodes are compatible if they are labeled with the same output and if they and their pairwise descendants have similar probability distributions of input-output pairs. More concretely, IOALERGIA checks whether the children reached with a common input follow the same probability distribution, which IOALERGIA checks with a test based on Hoeffding's bound. The compatibility check is performed recursively on their respective sub-trees.
- **merge(*red*, *blue*)**: Merging of *blue* into *red* node is performed in the same manner as in RPNI. However, when *blue* transitions (and its descendants) are recursively folded onto *red* transitions, the resulting transitions sum their frequencies.
- **toAutomaton(*red*)**: The final step of IOALERGIA encodes the IOFPT as an MDP. To do so, it normalizes the input-output transition frequencies and encodes them as probabilities. For example, the frequency sum of the input *button* from state q_2 in Figure 2.9 is 7, and 6 of those lead to the node labeled with *coffee*. Therefore, in the resulting model, *button/coffee* would have 85% (6/7) transition probability.

IOALERGIA example on the Mealy machine model of the coffee machine. Given a finite data set of discrete input-outputs, IOALERGIA starts by constructing an IOFPT. An example of an IOFPT can be seen at the left-hand side of Figure 2.9. IOALERGIA follows the red-blue framework, therefore the root node of the IOFPT is put into the *red* set, and its children into the *blue* set. In the next step IOALERGIA checks the compatibility of q_0 and q_1. These states are deemed compatible as they share the same state output (*init*) and their input-output frequencies follow the same distribution, as determined with test based on Hoeffding's bound. Since q_0 and q_1 are compatible, they are merged and we observe that the frequencies of the folded transitions are updated (right-hand side of Figure 2.9). The process then continues and results in a three-state automaton with the same structure as the one found in Figure 2.4. However, due to the limited size of the data set the computed transition probabilities from the state reached after executing *coin* are slightly different than the ones used for data generation. More concretely, from that state input *button* had 90% chance of getting *coffee* and 10% chance of resulting in *init*, while in the final learned model *button* produces a *coffee* in 85% (6/7) of the cases, while it produces *init* in the other 15% (1/7) of the cases.

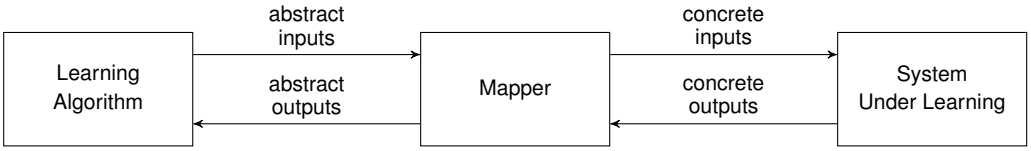


Figure 2.10: Alphabet abstraction with a mapper

2.4 Role of Abstractions in Automata Learning

We have established that automata learning algorithms work with discrete input and output data, and with relatively small input and output alphabets. In active automata learning, large input and output alphabets would lead to "practically infinite" learning, that is, the learning algorithm would always find new counterexamples or would be stuck in the closing loop. In the passive learning algorithms, large alphabets would often result in a model that is the same as the IOFPT, meaning that no states were merged and no meaningful structure was derived.

However, many real-world systems that lend themself quite well to automata learning, such as network protocols, do not necessarily have (reasonably sized) discrete input and output alphabets. For example, consider active learning of network protocols, such as MQTT. The number of valid actions is virtually unbounded. For example, a simple connection request can have an unbounded payload message, custom user ID, and various combinations of flags that define certain parameters of the connection. This is usually abstracted either to a single configuration (always using the same payload and flags) or to a set of configurations that fall into the same equivalence class.

Input and output alphabet abstraction in automata learning is done with a mapper component [1]. The mapper is placed "between" the learning algorithm and the SUL, as seen in Figure 2.10. It performs two opposite actions: abstraction and concretization. Concretization maps abstract input to concrete actions, that are then executed on the system. The output of concrete actions is then abstracted; that is, processed and mapped into one of the elements of the abstract output alphabet.

In the scope of this thesis, we examine abstraction methods that enable automata learning in the machine learning domain. More concretely, in Chapter 6 we investigate appropriate abstraction methods for RNNs, while in Chapter 9 we examine whether abstraction mappers can be automatically computed to enable automata learning in continuous stochastic environments.

2.5 Recurrent Neural Networks

An artificial neural network (ANN) consists of computational units, called neurons, that are connected via weighted edges. The neurons are usually organized into ordered layers with a dedicated input layer, a dedicated output layer, and intermediate layers, called hidden layers. ANNs can abstractly be viewed as functions $f(\mathbf{x}) = \mathbf{y}$ mapping an n-dimensional real-valued input $\mathbf{x} \in \mathbb{R}^n$ to an m-dimensional real-valued output $\mathbf{y} \in \mathbb{R}^m$. We consider a specific type of ANNs called recurrent neural networks. They can model sequential behavior through connections from one layer to itself or to a previous layer [282]. Through such connections, the output of an RNN depends not only on the current input, but also on the accumulated hidden state from computations involving inputs from previous time steps. This enables RNNs to model stateful reactive systems. The behavior of an RNN can be recursively defined as $f(\mathbf{x}_i, \mathbf{h}_i) = \mathbf{y}_i$ and $\mathbf{h}_i = g(\mathbf{x}_i, \mathbf{h}_{i-1})$, where $\mathbf{x}_i$, $\mathbf{y}_i$, $\mathbf{h}_i$ are the input, the output, and the hidden state vector at time step i, and $\mathbf{h}_0$ is the initial hidden state. There is a function f computing the RNN's output and a function g updating the RNN's state based on the current state and input. Having access to an internal memory makes RNNs ideal for challenging tasks on high-dimensional sequential data, such as natural language processing [232, 268], time series forecasting [67], and reinforcement learning in partially observable environments [92], where agents take actions based on a history of events.

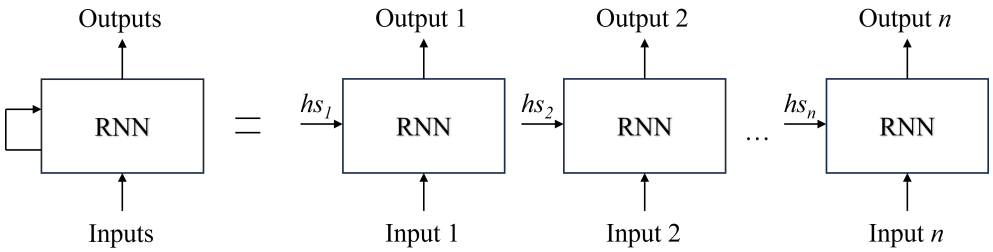


Figure 2.11: Visualization of RNN's sequence processing

Figure 2.11 shows an abstract demonstration of RNN sequence processing. The left-hand side of the figure contains a view in which an RNN receives a sequence of inputs, and produces a sequence of outputs. This mechanism is then "unrolled" on the right-hand side of the figure. We can observe that RNN has an initial hidden state hs_1 that is used to compute the output and the next hidden state hs_2 given the first element of the input sequence. The computed hidden state hs_2 is then used to compute the second output (and the third hidden state) based on the second element of the input sequence. This process is repeated until the whole sequence is processed, and we can observe that the output of the final element of the input sequence depends on the hidden state accumulated from previous steps. We further examine this process, and some assumptions made about the structure of the hidden state space in Chapter 6.

Throughout the thesis, we consider three types or RNNs, namely Elman RNNs [67], LSTMs [95], and GRUs [47]. In the remainder of this section, we outlined the mathematical definitions of these networks, following the PyTorch implementation notation [190]. We do not go into the specifics of the RNN training process, since the training process is not examined throughout the thesis, and we usually consider pre-trained RNNs.

2.5.1 Elman RNNs

Elman's RNN [67] (also referred to just as RNN) update equations can be expressed as follows:

$$h_t = g(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh}) \text{ where } g \in \{\tanh, \text{ReLU}\}$$
$$y_t = f(h_t W_{ho}^T)$$

The hidden state at time step t is denoted as h_t and the input at time step t as x_t. The activation function g which is usually either a hyperbolic tangent ($tanh$) or the Rectified Linear Unit (ReLU). Bias terms for the input-to-hidden and hidden-to-hidden layers are expressed with b_{ih} and b_{hh}, and W_{ih} and W_{hh} are weight matrices for the input-to-hidden and hidden-to-hidden connections. The output at time step t is represented with y_t, f is the activation function for the output layer, and W_{ho} is the weight matrix for the hidden-to-output connections.

The first equation updates the hidden state h_t based on the input x_t, the previous hidden state h_{t-1}, and their corresponding weights and biases. The activation function g is applied element-wise to the sum of the weighted inputs and biases. The second equation calculates the output y_t at time t based on the hidden state h_t using the weight matrix W_{ho}, and the activation function f is applied.

While Elman RNNs can capture dependencies in sequential data, they struggle with long-term dependencies due to the vanishing/exploding gradient problem [282]. The vanishing gradient is a phenomenon that occurs during the RNN training when gradients (that are propagated through several time steps) become very small or very large. Such gradients cannot sufficiently update the weights of the network, limiting the learning capabilities of the network, most notably the inability to learn long-term dependencies. However, LSTMs and GRUs introduce mechanisms to capture such dependencies efficiently.

2.5.2 Long Short-Term Memory

Long short-term memory networks (LSTM) [95] use memory cells and gating mechanisms that enable selective remembering or forgetting of information over long sequences. Memory cells consist of input, forget, and output gates. They regulate the flow of information in the memory cell, mitigating the vanishing gradient problem.

LSTM update equations can be expressed as follows:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + r_t * (W_{hg}h_{(t-1)} + b_{hg})) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t \odot c_{(t-1)} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

The input gate at timestep t is denoted as i_t, forget gate as f_t, cell gate as g_t, and the output gate o_t. They are used to control the flow of information into and out of the memory cell. σ represents the Sigmoid function, and $\odot$ is the Hadamard product.

The cell state c_t is updated based on the input gate i_t, the cell gate g_t, and the forget gate f_t. The cell state is then passed through a hyperbolic tangent function to produce the hidden state h_t, which is multiplied element-wise by the output gate o_t.

2.5.3 Gated Recurrent Units

Gated recurrent units (GRU) [47] RNN update equations can be expressed as follows:

$$
\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) \odot n_t + z_t \odot h_{(t-1)}
\end{aligned}
$$

In these equations, r_t are reset gates and z_t are update gates. They control the flow of information in the GRU cell. As with LSTMs, σ represents the sigmoid function.

The activation n_t is calculated based on the input x_t, the reset gate r_t, and the previous hidden state $h_{(t-1)}$. This activation is then used to update the hidden state h_t by interpolating between the activation n_t and the previous hidden state $h_{(t-1)}$ and it is weighted by the update gate z_t.

2.6 Basics of Reinforcement Learning

Reinforcement learning (RL) is a machine-learning paradigm in which an RL agent attempts to solve an RL task by interacting with the environment. Given the current state of the environment (in the context of RL also known as observation), the RL agent makes a decision and observes its consequences, that is a state change and a reward [216]. Rewards are used to define a task in RL. Positive rewards will "motivate" the agent to repeat the action, while negative rewards penalize the choice of said action from the current state. However, the agent must also consider possible future rewards when choosing an action from a given state. The expected future reward is a concept that prevents greedy action selection, that

is the RL agent is not concerned only with the optimality of the action in the current state, but with its future consequences.

Figure 2.12 abstractly shows the interaction process between an RL agent and an environment. An agent observes the current state of the environment through the state obs_t and based on this observation decides on an action a_t. Based on the action, the environment updates its current state, which the agents observe through the new observation obs_{t+1} and reward r_{t+1}.

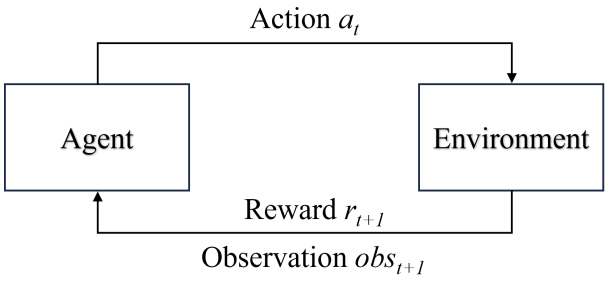


Figure 2.12: Interaction between a reinforcement learning agent and an environment

Reinforcement learning environments and tasks can be categorized in several ways [216, 221, 231]. In the following list, which follows the terminology and descriptions as outlined by Sutton and Barto [231], we present the main categorizations of RL settings and highlight those used throughout the thesis.

- **Episode Length.** RL tasks can either be episodic or continuous. In episodic RL, tasks are executed in episodes of finite length, with well-defined starting and stopping criteria, such as achievement of a goal or reaching a terminal state (eg. reaching a certain "bad" state or maximum episode length). Alternatively, in continuous RL tasks do not have a terminal state, that is,the agent interacts with the environment indefinitely. Throughout the thesis, we investigate episodic RL tasks.

- **Model Availability.** We can distinguish between model-free and model-based RL. In model-based RL, an agent either builds or is given a model of the environment, and it can simulate future environment dynamics (states and rewards) and plan accordingly. Alternatively, model-free approaches do not use (or explicitly construct) the model of the environment. They instead learn the value functions of states through trial and error. In the thesis, we consider model-free RL approaches.

- **Action Space.** Actions that an agent can take can either be discrete or continuous. Discrete actions are chosen from a finite set of possible actions, while a continuous action space is usually represented as a set of real-valued vectors. In the thesis, we consider only discrete action spaces.

- **Observation Space.** Similarly to the action spaces, the observation space of the environment can be discrete or continuous. In this thesis, we examine environments with both discrete and continuous observation spaces.

- **State Observability.** States in the RL can either be fully or partially observable. In a fully observable environment, the current state of the environment is uniquely identified by its observation, while in the presence of partial observability, the observations seen by the agent do not fully identify the state. Partial observability increases the complexity of reinforcement learning tasks, as an agent has to have some kind of mechanism that helps him identify or approximate the current state from incomplete observations. In our work, we examine both fully and partially observable settings.

2.6.1 Formalization of Reinforcement Learning

Reinforcement learning tasks can be formalized as follows:

- Environments behave as an MDP $\langle Q, I, O, \delta, \lambda, q_0 \rangle$ (see Section 2.1.2)
- A reward function $R : Q \to \mathbb{R}$ defines the environment's reward structure
- The agents attempt to find a policy π such that $\pi = \pi_{\text{argmax}} \sum_{t=0}^{T} \gamma^t R_t$, where
  - Policy $\pi : Q \to I$ is a mapping from states to actions
  - $\gamma \in [0, 1)$ is a discount factor, which determines the agent's preference for immediate rewards compared to the expected future reward
  - $T \in \mathbb{N}$ is the episode horizon (time steps)
  - R_t is the reward at time step t

We further define a state-value function $V^\pi(q)$ that represents the expected cumulative reward that the agent can achieve from a given state q under policy π. Formally, $V^\pi(q) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| q_0 = q\right]$, where $\mathbb{E}_\pi$ denotes the expectation over trajectories generated by policy π.

Since an agent makes an action from a state, we define a state-action value function $Q^\pi(s, a)$, such that $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| q_0 = s, a_0 = a\right]$, where $\mathbb{E}_\pi$ denotes the expectation over trajectories generated by policy π, starting from state s and taking action a. With this formalization in mind, we can also formalize the goal of RL as search for policy $\pi = \text{argmax}\, Q^\pi(s, a)$, that is, as a search for a policy that selects an optimal action in each state of the environment. Computation or approximation of the Q-function is a central concept behind several influential RL algorithms [154, 269].

This formalization assumes full state observability, and therefore an observation fully identifies a state of the environment. In this setting, states have the Markov property, ensuring the existence of an optimal memoryless policy π. In such policies, history does not affect the selection of actions in the current state. In Chapter 7 we further reason about policies under the presence of partial observability, where states do not maintain the Markov property.

We do not go into more detail about RL at this moment, but introduce the necessary RL concepts at the appropriate places throughout the thesis. More concretely, in Chapter 7 we introduce tabular reinforcement learning methods [269] that can solve RL tasks in the fully observable domain with discrete observations and output, and show how they can be extended with the help of automata learning to solve such tasks in the presence of partial observability. In Chapter 9 we briefly outline several Deep-RL learning methods and show how automata learning can be used to test them with the learned models of continuous stochastic environments.

3

Design and Development of an Automata Learning Framework

Declaration of Sources

This chapter is based on our work on AALpy, an automata learning library written in Python. AALpy was presented in two publications: "AALpy: Active Automata Learning Library", which was presented at ATVA 2021 [164], and appeared in 2022 version of NASA Journal Innovations in Systems and Software Engineering [166].

3.1 Motivation

Software libraries enable modern software development processes. In the academic setting, they enable transparent research as they provide researchers with standardized algorithm implementations and enable fair comparison of new algorithms with existing baselines. In recent years, the automata learning ecosystem largely relied on LearnLib [111], and its influence on the automata learning landscape cannot be overstated. However, we identified a need for an automata learning library native to the Python ecosystem, that would enable simple and efficient applications of automata learning in various domains.

Therefore, we developed AALpy, an automata learning library written in Python. AALpy is hosted on Github [1] and comes with a permissive MIT licence. Since its inception, the target audience of AALpy were not only automata learning researchers but a broader public that might be interested in the automatic modeling of black-box systems. At the time of AALpy's inception, Python has just become the most popular programming language according to the TIOBE index, and has maintained that position ever since [244]. Popular and influential software machine-learning libraries, like Keras [48] and PyTorch [190], mainly provide Python APIs, and the Python ecosystem provides a vast amount of libraries that could be used to interface to other black-box systems. An example of such a library would be Scapy [217], a package manipulation library that can be used on various networked systems.

AALpy served as a central hub for the majority of work performed during my doctoral studies: my work either was on AALpy itself, or on the use of automata learning through AALpy. At the time of the writing, AALpy has had 11 contributors, 20 forks, and was used in multiple open-source projects unrelated to this thesis. Some interesting use cases that relied on AALpy are outlined in Section 3.9.

[1] https://github.com/DES-Lab/AALpy

In the remainder of this chapter, we outline the AALPY's design principles, describe its capabilities, evaluate the implemented learning algorithms, and outline AALPY's position in the automata learning landscape.

3.2 Design Philosophy

Before we describe AALPY in more detail, we outline the design philosophy used throughout the development process. The following key aspects define AALPY:

- **Simplicity and Ease of Use**: When designing AALPY, we put significant effort into simplifying the automata learning process for an end user. This does not entail that AALPY is limited in any way, but that an end user should be able to use AALPY's full functionality with a relatively simple workflow. AALPY's target audience is not only existing practitioners of automata learning but any person who might want to learn the stateful behavior of some black-box system. Therefore, the design of AALPY enables users to use automata learning without strong background knowledge about the principles of automata learning, while the advanced users can still extend AALPY's functionalities to suit their needs.
- **All-encompassing Automata Learning Solution**: Automata learning algorithms do not exist in a vacuum. They require implementations of various types of finite state models, and the output of the automata learning algorithms, that is learned models, are often used for other purposes such as model checking. Therefore, we made AALPY a streamlined solution for automata learning: it features an automata submodule, many auxiliary functions that a user might need, as well as a model-checking submodule that interfaces to popular model checkers.
- **Availability of Examples**: The previous points are extended with strong documentation. AALPY is documented via extensively commented code, examples that cover all of its functionality[2], and a Wiki[3] that covers many practical aspects of automata learning.
- **Efficiency**: AALPY's implementations of learning algorithms are optimized with respect to runtime and memory usage. In the case of active automata learning, AALPY also optimizes the learning process by minimizing the required interaction with the SUL through various heuristics and counterexample processing strategies.
- **Reproducibility**: All experiments in AALPY are reproducible with the definition of a random seed at the beginning of the Python script. Results might not be fully reproducible during active learning of non-simulated stochastic systems, as their randomness is not dependent on Python's random number generator.
- **Compatability**: Since Python serves as an interface language to many systems, we designed AALPY with minimal overhead. For example, AALPY has a single dependency and is compatible with Python $\geq$ 3.6. This enables the integration of AALPY with the vast majority of Python libraries. We also aim to ensure backward compatibility between AALPY's bi-monthly releases.

3.3 Library Structure

AALPY's structure can be divided into four distinct modules. These modules can be seen in Figure 3.1, and they work together to enable a seamless automata learning process. Those modules are:

- **Automata Module**: Almost all functions in AALPY directly or indirectly work on automata. Therefore, AALPY features implementations of all modeling formalism outlined in Table 3.1. Automata implementations are supported by many helper functions that are most commonly used in automata learning, e.g., computation of the characterization set, computation of minimal state prefixes, and computation of the shortest path between two states.

[2] https://github.com/DES-Lab/AALpy/blob/master/Examples.py
[3] https://github.com/DES-Lab/AALpy/wiki

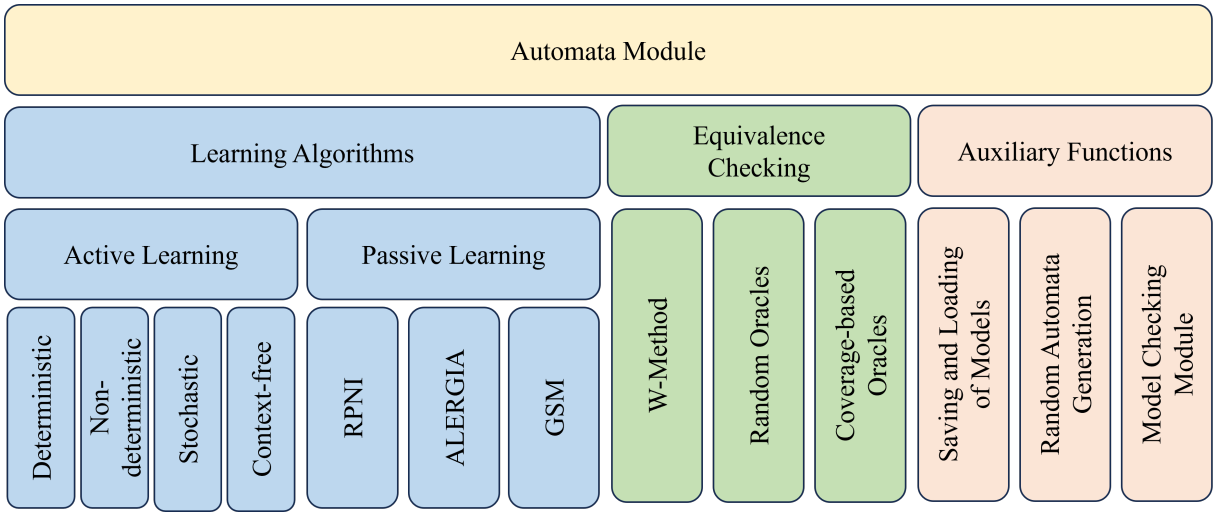


Figure 3.1: Abstract view of AALPY's modules

- **Learning Algorithms**: The core of AALPY are its learning algorithms. They are divided into active and passive learning sub-modules. In the remainder of this chapter, we outline all available algorithms, and briefly explain the implemented optimization heuristics.
- **Equivalence Oracles**: AALPY offers a wide selection of equivalence oracles that aim to ensure the correctness and efficiency of the active learning process. However, we choose to display equivalence oracles as a separate module from active automata learning. While one might argue that they should be categorized as part of the active automata learning module, we consider them as separate modules as they might be used independently of active learning algorithms. For example, they could be used for the generation of a model-based test suite independent of the automata learning process.
- **Auxiliary Functions**: The following are some helper functions that make AALPY an all-encompassing automata learning solution: random automata generation for all supported formalism, saving and loading of models that conform to the .dot format [177], export of stochastic models to PRISM [128] format, simple statistical model checking, and visualization of learned models.

Another view of AALPY's functionality is offered in Table 3.1. In this table, we provide a more detailed view of supported modeling formalisms and of the implemented algorithms.

Table 3.1: Supported formalisms, learning algorithms, and a selection of AALPY's features

Automata Type	Supported Formalisms	Algorithms	Features
Deterministic	DFAs Mealy Machines Moore Machines	L^* KV RPNI	Seamless Caching 6 Cex. Processing Strategies 13 Equivalence Oracles
Non-Deterministic	ONFSM Abstracted ONFSM	L^*_{ONFSM}	Size Reduction though Abstraction
Stochastic	Markov Chains Markov Decision Processes Stochastic Mealy Machines	L^*_{MDP} L^*_{SMM} ALERGIA IOALERGIA	3 Cex. Processing Stategies Exportable to PRISM Statistical Model Checking
Context-Free	VPDA/1-SEVPA	KV_{VPA}	Specification of exclusive call-return pairs

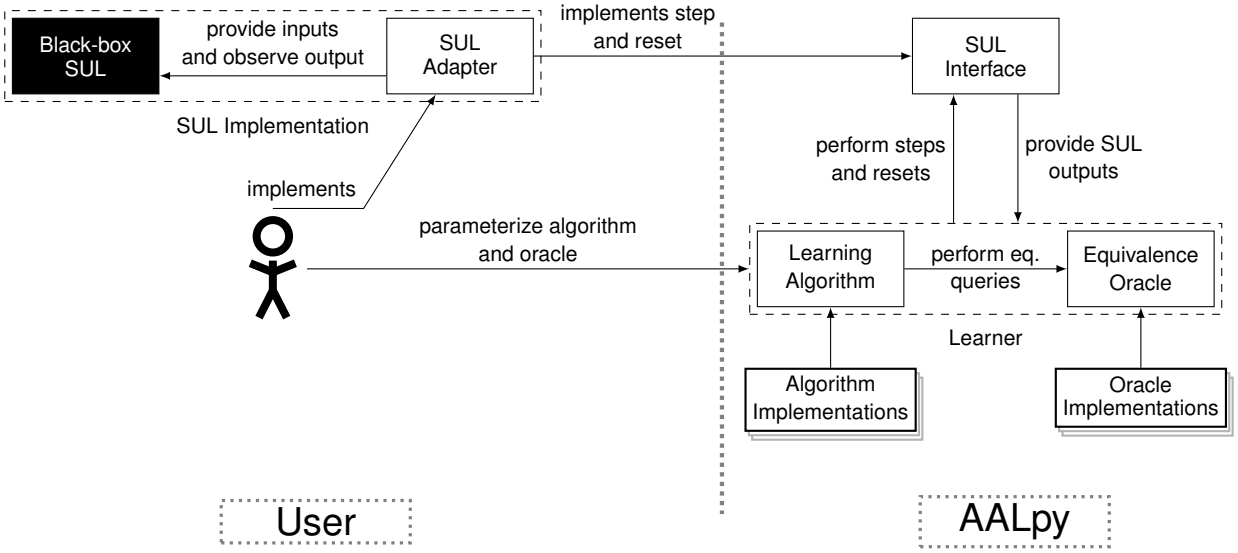


Figure 3.2: AALPY's active automata learning interface and structure

3.4 Active Learning Algorithms

In this section, we outline all active learning algorithms supported in AALPY version 1.4.1. We briefly describe each algorithm, and outline the heuristics used to improve its efficiency. All active automata learning algorithms in AALPY are used with the following three-step process:

1. Implement the `SUL` interface
2. Select and parameterize an equivalence oracle
3. Run the learning algorithm

Implement the `SUL` interface. When learning models that capture the input-output behavior of some black-box system, we need to implement a `SUL` interface that defines the interaction between the learning algorithm and the SUL. The `SUL` interface contains three methods that a user needs to implement: `pre`, `post`, and `step`. The methods `pre` and `post` are used to reset a SUL, that is `pre` is called before each input query, while `post` is called after an input query. The separation of reset into `pre` and `post` was inspired by LearnLib [111]: with `pre`, we initialize and setup the SUL, while `post` shall support a graceful shutdown and memory cleanup. Note that often defining just `pre` is sufficient. Once the SUL has been reset, an input query is executed. An input query is a sequence of steps on the SUL, so the `step` methods define an execution of a single step on the SUL. It maps an element of an input alphabet to a concrete action, executes it on the SUL, and processes the output observed. Therefore, the implementation of `step` also serves as a mapper component, as described in Section 2.10. Implementation of the SUL interface and the definition of a proper abstraction level is the most labor-intensive part of active automata learning. An implementation of a single `SUL` interface can be used with all active learning algorithms, enabling effortless switching between learning formalisms.

Select and parameterize an equivalence oracle. Once the SUL interface has been defined, we need to select one of the 13 available equivalence oracles, which will be used during learning to answer an equivalence query. Selected oracles can then be parameterized to control the amount and kind of testing used to check black-box observational equivalence. Alternatively, the user might choose to implement a custom oracle, which is enabled with the `Oracle` interface. Available oracles are described in more detail in Section 3.6.

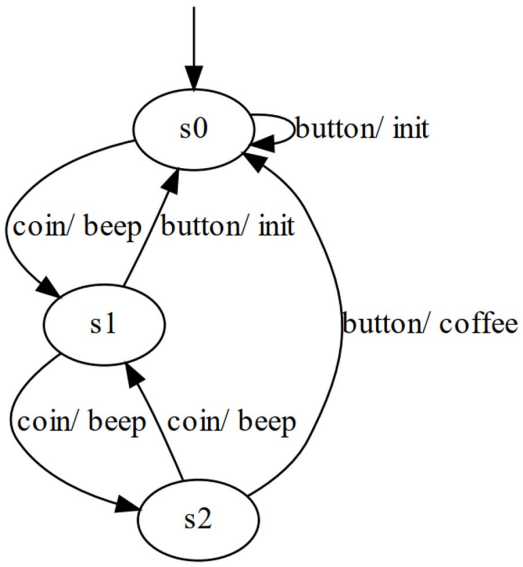
Run the learning algorithm. The final step is the call of the learning algorithm itself. Calls to all learning algorithms follow the same basic structure, so the user can easily switch between different algorithms and paradigms (such as deterministic and stochastic learning). Default parameterizations of all learning algorithms provide a good basis for learning, and an advanced user can further adapt certain aspects of learning with additional parameters. The execution of the learning algorithm returns the learned model and outputs basic learning statistics, such as the total number of input queries, number of learning steps, and learning time.

This process is abstractly represented in the Figure 3.2. This figure also abstractly outlines AALPY's active automata learning structure.

Listing 3.1: Learning a model of a coffee machine with AALPY

```
from aalpy.SULs import SUL
from aalpy.oracles import RandomWordEqOracle
from aalpy.learning_algs import run_Lstar, run_stochastic_Lstar

class CoffeeMachineSUL(SUL):
    def __init__(self, coffee_machine):
        super().__init__()
        self.coffee_machine = coffee_machine

    # reset the coffee machine to an initial state
    def pre(self):
        self.coffee_machine.empty()
        self.coffee_machine.num_coins = 0

    def post(self):
        pass

    # define a mapping from abstract inputs (string) to calls on the SUL
    def step(self, letter):
        if letter == 'coin':
            return self.coffee_machine.add_coin(1)
        else:
            return self.coffee_machine.press_button()

# black-box coffee machine whose interface is public
coffee_machine = CoffeeMachine()

# pass the black box to the SUL interface used by the learning algorithm
sul = CoffeeMachineSUL(coffee_machine)

# define the input alphabet
input_alphabet = ['coin', 'button']

# choose and parameterize an equivalence oracle
eq_oracle = RandomWordEqOracle(input_alphabet, sul, num_walks=200,
                               reset_after_cex=True)
# run the learning algorithm
learned_model = run_Lstar(input_alphabet, sul, eq_oracle, automaton_type='mealy')

# if we observed stochastic behavior, we could switch to stochastic learning
# learned_stochastic_model = run_stochastic_Lstar(input_alphabet, sul, eq_oracle,
#                                                 automaton_type='mdp')

learned_model.visualize()
```

Active Learning Code Example. In Listing 3.1 we demonstrate the three-step active learning setup. Suppose we are given an interface to the coffee machine examples outlined in the previous section, and we can interact with it through the `CoffeeMachine` class. We define a `CoffeeMachineSUL` in




```
Hypothesis 1: 1 states.
Hypothesis 2: 3 states.
----------------------------------
Learning Finished.
Learning Rounds:  2
Number of states: 3
Time (in seconds)
  Total                : 0.02
  Learning algorithm   : 0.0
  Conformance checking : 0.02
Learning Algorithm
 # Membership Queries  : 17
 # MQ Saved by Caching : 9
 # Steps               : 71
Equivalence Query
 # Membership Queries  : 201
 # Steps               : 4038
----------------------------------
Visualization started in the background thread.
Model saved to LearnedModel.pdf.
```

Figure 3.3: Outcome of Listing 3.1

lines 5-23. In the `pre` method we reset the coffee machine by calling empty and setting the number of coins entered to 0, and in the `step` method we define a mapping between elements of the input alphabet to function calls of the `CoffeeMachine` class. In this simple example, outputs received from the coffee machine are sufficiently abstract so we do not need to process them. In Line 32 we define an input alphabet used throughout the learning. In this example, the input alphabet covers the whole functionality of the SUL, but in some learning scenarios, we might be interested in only a subset of the whole alphabet. We then select one of the equivalence oracles in Line 35 and run the learning algorithm with the default parameterization in Line 38. The learned model is then visualized in Line 44. Suppose that during execution we observed that the SUL behaves non-deterministically. We then might switch to learning of non-deterministic or stochastic models by simply replacing the algorithm call with another function, as shown in Line 41. The outcome of this code example can be seen in Figure 3.3.

3.4.1 Active Learning of Deterministic Models

AALPY provides efficient implementations of L^* [14] and KV [116]. Both algorithms can be used to learn DFAs, Mealy machines, and Moore machines, and support all deterministic counterexample processing strategies outlined later in this section. Both algorithms cache all input-output traces observed during learning and conformance checking, greatly reducing the amount of redundant interactions with the SUL. These algorithms are used to learn models of deterministic systems, and if a non-deterministic behavior is observed during learning or conformance checking, the algorithm will throw an exception and output a trace that caused non-deterministic behavior.

Given a sufficiently strong equivalence oracle, both algorithms will yield the same models of the SUL. However, the learning process of L^* and *KV* differs. In general, L^* asks more input queries during the learning process, but it is less dependent on the equivalence oracle. On the other hand, *KV* is more efficient during learning, but mostly identifies a single state with each counterexample.This makes *KV* more reliant on the ability of the equivalence oracle to explore all parts of the SUL, while L^* explores parts of the SUL on its own. Abstractly we could view L^* as a breadth-first exploration of the SUL's behaviour, and *KV* as a depth-first exploration. For example, when learning a 1000-state DFA, L^* might learn it in just 3 learning rounds. This means that the equivalence oracle returned two counterexamples, and in the third learning round, it could not find a new counterexample, leading to

halting the learning process. *KV* will require 999 learning rounds, where an equivalence oracle will have to find 998 counterexamples. The choice of the appropriate learning algorithm for the SUL is left to the user, but in general, we recommend the usage of the *KV* learning algorithm as it is generally more efficient. However, if testing resources are limited, the breadth-first exploratory nature of L^* could provide higher confidence in the correctness of the learned model.

L* implementation details

AALPY provides a highly customizable implementation of the L^* algorithm. The default parameterization follows the L^* algorithm described in Section 2.2.3. An example usage of L^* with AALPY can be seen in Listing 3.1. The following listing outlines some of the available customization options:

- **Counterexample processing strategy**: We further elaborate on available counterexample processing strategies in Section 3.4.5.
- **Closing strategy**: This strategy selects the row that will be closed if more than one row in the observation table are the cause of a closeness violation.
- **Provide samples**: The user can provide pre-recorded input-output samples generated by the SUL to the algorithm, which are then added to the cache. This could reduce the total amount of interaction with the SUL.
- **Max. learning rounds**: If the maximum number of learning rounds is defined, and the learning process reaches the defined learning round the learning will stop. This is useful when the SUL's behavior cannot be captured with regular languages, or when we do not necessarily want to learn the whole behavior of the SUL.
- **Suffix-closed E set**: Controls whether all suffixes of the processed counterexample will be added to the observation table. If set to False, only the distinguishing suffix will be added, and the learning algorithm will ensure that the observation table is semantically suffix closed [228].
- **Print Level**: The user can choose the amount of information displayed during the active automata learning process. By default, AALPY will continuously print the current learning round, and at the end of the learning process, it will display learning statistics. An example of such output could be seen in Figure 3.3.

KV implementation details

AALPY's implementation of *KV* is very efficient with respect to the number of learning steps, that is, on average our implementation of *KV* requires fewer interactions with the SUL than L^*.

AALPY's *KV* implementation is efficient due to its counterexamples processing strategy, which is consistent with the one used in the observation pack algorithm [99]. The implemented counterexample processing strategy by default asks fewer input queries than the default *KV* counterexample processing, but more importantly, it ensures that throughout learning shortest prefixes are used to identify states in the classification tree. A classification tree is a data structure maintained by *KV*, and it is used to identify states and transitions during the learning process.

More precisely, once a counterexample *cex* is returned by an equivalence oracle, a suffix-based counterexample processing strategy like *RS* splits it into v, a, and w, such that $v \cdot a \cdot w = cex$. Let *getPrefix(seq, hyp)* be a function that returns a prefix of a hypothesis *hyp* state reached after executing *seq*. The prefix of the hypothesis state reached after executing $v \cdot a$, that is *getprefix($v \cdot a$, hyp)* identifies a state in the hypothesis (that corresponds to the classification tree leaf) which is split with the discriminator w. The access string of a new state (leaf) is defined with *getPrefix(v, hyp)* $\cdot\ a$. This counterexample processing strategy works with all suffix-based counterexample processing strategies implemented in AALPY, and when used with *RS* counterexample processing it ensures that the shortest prefixes will be added to the classification tree as access sequences of its nodes. This in turn greatly decreases query length throughout the learning process.

3.4.2 Active Learning of Non-deterministic Models

When a SUL produces outputs non-deterministically we can learn an observable non-deterministic finite-state machine (ONFSM) with L^*_{ONFSM}. ONFSMs could be seen as a generalization of Mealy machines with allowed non-deterministic behavior. More precisely, from each state, a single element of the input alphabet could map to multiple elements of the output alphabet. However, each input-output pair from a given state has to define a unique transition, therefore satisfying the *observable* non-deterministic property, akin to deterministically labeled MDPs [66].

AALPY's implementation mostly follows the original algorithm outlined in [66]. However, we relaxed the strict all-weather assumption made in the original paper and approached learning non-deterministic models from a testing perspective.

In the original paper, it is assumed that the SUL returns all possible continuations for a given suffix. We do not make this assumption but rather assume that with sufficient testing (repeat each input query n times) we will observe all possible outputs. Furthermore, in each learning round, we recompute an observation table to be consistent with all previously seen observations. This design decision makes the algorithm more efficient (as we can use lower n to sufficiently approximate the all-weather assumption) and does not entail additional interaction with the SUL, since we are using a cache to repopulate the observation table. This design decision was inspired by L^*_{MDP} and it enables the reuse of the same SUL implementation for deterministic, non-deterministic, and stochastic learning.

AALPY also features an extension [195] of the classic ONFSM learning algorithm. This extension learns abstracted ONFSM by introducing additional equivalence classes for outputs. This abstraction mechanism enables the creation of smaller models and can be used to learn non-deterministic representations of deterministic systems with large input alphabets.

3.4.3 Active Learning of Stochastic Models

AALPY features implementations of original and improved versions of L^*_{MDP}, as well as L^*_{SMM}, an adaptation of the improved algorithm to stochastic Mealy machines. We examine these stochastic active learning algorithms in great detail in Chapter 4.

3.4.4 Active Learning of Context-free Languages

The most recent addition to AALPY is the implementation of a deterministic visibly pushdown automata (VPDA) automata learning algorithm. From a practical perspective, the invocation of VPDA learning follows the same three stage process as the other active learning algorithm, with the addition of an explicit definition of call-return pairs. That is, in this learning paradigm, a user has to define which elements of the input alphabet push to the stack, and which pop from it.

The implementation closely follows the algorithm described in the fourth chapter of Malte Isberner's PhD thesis [106]. The adaptation of the presented algorithm to *KV* was straightforward, as the original VPDA learning was presented as an abstract extension of existing tree-based automata learning algorithms, namely TTT [109] and observation pack [99]. We have slightly extended the algorithm by adding two heuristics: early identification of sink-states and the ability to define exclusive call-return pairs.

3.4.5 Counterexample Processing Strategies

All deterministic algorithms in AALPY can use the same counterexample processing strategy suite as LearnLib [108, 111]. In the following list, we briefly outline deterministic counterexample processing:

- **Native Counterexample Processing**: Both L^* and *KV* can be used as described in their original papers [14, 116]. However, those counterexample processing strategies are sub-optimal as they cause resource-intensive learning.

- **RS**: Rivest-Schapire is the default counterexample processing strategy for both algorithms. It was described in more detail in Section 2.2.3.
- **Linear Counterexample Processing**: Linear counterexample processing searches for a distinguishing suffix by linearly processing the counterexample. The counterexample can be processed linearly in both directions, that is starting from the front or the back of the counterexample.
- **Exponential Counterexample Processing**: Exponential search is used to limit the range of *RS* counterexample processing. In *RS*, the lower and upper bound is initialized to the beginning and at the end of the counterexample, and with exponential search we can limit the *RS* bounds. Like in linear counterexample processing, a search for the distinguishing suffix can be performed in both directions.
- **Longest Prefix**: As observed by Shabaz and Groz [220], a counterexample could be split into a prefix and a distinguishing suffix by simply removing the longest prefix found in the observation table from the counterexample. Unlike *RS*, this counterexample processing strategy does not require additional input queries, but the learning algorithms with this counterexample processing are usually less efficient than ones with *RS*, since the found distinguishing suffixes are longer.

The non-deterministic learning algorithm uses the *Longest prefix* counterexample processing strategy, and stochastic active learning algorithms use adapted versions of *RS* and *Longest prefix* counterexample processing strategies. Counterexample processing strategies used in active stochastic learning are further discussed in Chapter 4.

3.5 Passive Learning Algorithms

While AALPY was initially developed as an active automata learning library, we quickly realized the importance of having efficient implementations of passive learning algorithms. Therefore, we extended AALPY with passive learning algorithms that can learn deterministic and stochastic models, namely RPNI and IOALERGIA.

3.5.1 Passive Learning of Deterministic Models

AALPY provides an efficient implementation of the RPNI [58, 187] algorithm that can be used to learn DFAs, Mealy machines, and Moore machines consistent with the provided dataset. Our approach for learning of Moore machines differs from passive Moore machine learning presented in [81]. We view Moore machines as a generalization of DFAs, that is they share the same structure and properties and only differ in their output mapping: a DFA is restricted to the Boolean domain, while the output domain in Moore machines is a discrete set of labels. This view allows us to trivially adapt RPNI to Moore machine learning by simply "lifting" the output domain. The same view on Moore machine learning is taken in our L^* and *KV* implementations.

Listing 3.2 shows the invocation of the RPNI algorithm as well as the data input format. Note that for RPNI, the data set does not necessarily have to be prefix-closed. That is, we do not require an output label for each input of an input sequence, only for the last output. Examples of such sequences are seen in lines 6-12. The RPNI algorithm is run with a single call to the `run_RPNI` function, as shown in Line 16. The algorithms return the model consistent with the provided data, and outputs the total algorithm runtime.

3.5.2 Passive Learning of Stochastic Models

AALPY offers implementations of ALERGIA [41] and IOALERGIA consistent with the description of the original IOALERGIA algorithm proposed by Mao et al. [144]. For conciseness, we refer to both algorithms simply as ALERGIA, since they share the same code basis, and only differ in the fact that

Listing 3.2: Example of passive stochastic model learning with RPNI

```
1
2  from aalpy.learning_algs import run_RPNI
3
4  # RPNI data is a list of sequence-label pairs
5  data = [
6      (['button', 'coin', 'button', 'coin'], ' beep'),
7      (['button', 'coin', 'button'], ' init'),
8      (['button', 'button', 'coin', 'button'], ' init'),
9      (['button', 'coin', 'coin', 'coin', 'coin', 'button'], ' coffee'),
10     (['coin', 'coin', 'coin', 'coin', 'coin', 'coin'], ' beep'),
11     # ...
12     (['coin', 'coin', 'coin', 'button', 'coin', 'coin', 'button'], ' coffee')
13 ]
14
15 # run RPNI with the provided data
16 learned_model = run_RPNI(data, automaton_type='mealy', print_info=True)
17 # visualize the learned model
18 learned_model.visualize()
```

ALERGIA is used to learn Markov chains, while IOALERGIA can learn MDPs and stochastic Mealy machines.

As described by Mao et al. [144], ALERGIA could be implemented to optimize for memory or for accuracy. When optimizing for memory, ALERGIA uses a single IOFPT and performs compatibility checks on frequency values that are continuously updated during state merging. This results in models that are not fully consistent with the provided dataset, since the merging of transition frequencies slightly skews the input-output frequency distribution with respect to the input-output frequency distribution of the provided dataset. On the other hand, the accuracy-optimized version of ALERGIA maintains formal correctness guarantees. In this version, the algorithm maintains a mutable and immutable IOFPT. The mutable tree is used throughout the algorithm to merge nodes and subsequently update input-output frequencies, while the input-output frequencies present in the immutable tree are used during the compatibility check. This introduces significant memory overhead since large datasets entail big IOFPTs. However, in AALPY we provided, to the best of our knowledge, the first ALERGIA implementation that is optimized for accuracy while only maintaining a single IOFPT. This is achieved by performing all merges on a single tree, but still maintaining the immutable records of the original input-output frequencies and original children in each node. This way a single IOFPT remains constant in size throughout the algorithm execution, as the *red* nodes are "overlayed" on top of it.

Listing 3.3 shows how ALERGIA can be called from AALPY to learn MDPs. Note that unlike for RPNI, provided data sequences are prefix-closed, that is, outputs are defined for all elements of the input sequence. An example of such a dataset can be seen in lines 4-16. Note that MDP traces start with an initial output as defined in Section 2.1.2. In Line 19 ALERGIA is called and an MDP that captures the input-output behavior of the provided dataset will be returned. The exact syntax of the input data format is defined in AALPY's Wiki and in the RUN_ALERGIA function header comments.

3.5.3 Active-Passive Automata Learning

AALPY provides an implementation of an iteratively refined passive learning [263], which we dub active-passive learning[4]. In this framework, passive learning algorithms are extended with the active sampling of the SUL. That is, the active-passive algorithm iteratively extends the dataset used for passive automata learning according to some sampling strategy. In the current form, a user simply connects an implementation of a `Sampler` abstract class that is used to define a sampling strategy to the learning algorithm,

[4]While active-passive is an oxymoron, in this context it concisely refers to the fact that the passive algorithms are extended with active sampling.

Listing 3.3: Example of passive stochastic model learning with ALERGIA

```
from aalpy import run_Alergia

data = [
    [' init', ('button', ' init'), ('button', ' init')],
    [' init', ('coin', ' beep'), ('button', ' init')],
    [' init', ('coin', ' beep'), ('button', ' coffee')],
    [' init', ('button', ' init'), ('coin', ' beep')],
    [' init', ('coin', ' beep'), ('coin', ' beep'), ('coin', ' beep')],
    [' init', ('coin', ' beep'), ('coin', ' beep'), ('coin', ' beep')],
    [' init', ('button', ' init'), ('coin', ' beep'), ('coin', ' beep')],
    [' init', ('coin', ' beep'), ('coin', ' beep'), ('coin', ' beep')],
    [' init', ('coin', ' beep'), ('coin', ' beep'), ('button', ' init')],
    # ...
    [' init', ('button', ' init'), ('button', ' init'), ('coin', ' beep')],
]

# run alergia with the data and automaton_type set to 'mdp' to learn a MDP
learned_model = run_Alergia(data, automaton_type='mdp', print_info=True)
# visualize the learned model
learned_model.visualize()
```

and the dataset used for passive learning is iteratively extended with obtained samples. This framework could be trivially adapted to use equivalence oracles instead of the `Sampler` class, that is, the dataset would be extended with the counterexamples that would ensure that the subsequent iterations of the active-learning process will be more consistent with the SUL.

3.5.4 Generalized State Merging Framework

AALPY features a generalized state merging (GSM) framework based on the red-blue passive automata learning framework. GSM provides AALPY users with the ability to easily extend existing passive learning algorithms with custom merging and node compatibility operators. For example, in GSM the user might extend existing RPNI implementation with evidence-driven state merging [130]. The GSM framework present in AALPY offers similar functionalities as *flexfringe* [255]. We do not report on GSM in more detail, since the work on GSM is mostly done by Benjamin von Berg in the scope of his PhD thesis. Benjamin will continuously work on GSM and describe the GSM functionalities in upcoming publications.

3.6 Equivalence Oracles

The quality of the learned models in the active learning paradigm heavily depends on the ability of the equivalence oracle to find all counterexamples. We have emphasized the equivalence of oracle development, as to enable reliable automata learning.

As outlined in Sect 2.2.2, we consider three types of equivalence oracles: W-Method, purely random oracles, and oracles that provide some notion of coverage. In the remainder of this section, we outline the working principles of selected equivalence oracles. Note that all oracles return a counterexample as soon as the execution of a test case reveals a difference between the SUL and the hypothesis.

W-Method. AALPY's implementation of the W-Method equivalence oracle closely follows Alg. 2.3. AALPY also offers a breath-first exploration oracle that fully explores the state space up to a certain depth. While the breadth-first oracle does not offer practical benefits compared to the W-Method, we consider it as a useful inclusion when using AALPY in an educational setting.

Random Oracles. AALPY features implementations of random walk and random word equivalence oracles. Random word equivalence oracle follows Alg. 2.2, while the random word oracle determines the length of the test cases according to the stopping probability p. In both oracles the user defines the total number of test cases, parameters to determine the test-case lengths, and whether the test-case counter should be reset after every counterexample.

We also provide a PAC [252] version of the random word oracle. It follows the same working principle as the random word oracle, with the difference that the total number of test cases is defined as $\frac{1}{\varepsilon} \cdot \left(\log_e\left(\frac{1}{\delta}\right) + lr \cdot \log_e(2)\right)$, where ε is the generalization error, 1 - δ is the confidence, and *lr* is the current learning round. This formulation makes the returned hypothesis an ε-approximation of the correct hypothesis with the probability of at least 1 - δ [156].

Random oracles can be used for all types of active automata learning, that is, for deterministic, non-deterministic, stochastic, and context-free learning. They do not exploit the model structure during test-case generation, and as such are a natural choice for testing environments with non-deterministic or stochastic choice of outputs with respect to the input. While some other model-guided oracles could in theory be adapted to other formalisms aside from deterministic learning, we observed that random testing is often sufficient to find a sufficient amount of counterexamples in such scenarios. We report on these findings, in the context of active stochastic model learning, in Chapter 4.

Model-guided Oracles. In the following list we briefly describes the selection of implemented model-based oracles:

- Random-W Method: AALPY's implementation of the Random-W Method closely follows Alg. 2.3. Each state of the hypothesis serves as an origin for a test case, followed by a random walk and a randomly sampled element of the characterization set. This oracle ensures state coverage, as each state is used as an origin for the test case exactly n times, and by the addition of an element of the characterization set potentially explores otherwise hard-to-reach areas of the SUL.
- Transition Focus Oracle: This equivalence oracle focuses either on self-loops or transitions that lead to different states. This equivalence oracle could be used on systems that behave similarly to balanced parentheses grammars. In such systems, interesting behavior usually occurs on the transitions between states, and potential counterexamples are usually found by focusing on these transitions.
- k-way State Coverage Oracle: This oracle computes a test suite that contains a test case that covers every k-combination or k-permutation of states with additional random walks at the end.
- k-way Transition Coverage Oracle: This oracle selects test cases based on random testing and optimizing the k-way transition coverage of the hypothesis. The oracle follows a two-step process, in which it first generates a large number of random walks. In the second step, it greedily selects a subset of these tests to optimize coverage.

Additional Equivalence Oracle Features. Users have the option of implementing custom equivalence oracles through the `Oracle` abstract class. Given that sometimes a user might have a hand-crafted set of test cases that he would like to use during equivalence checking in conjunction with the equivalence oracle, each oracle can also be instantiated with a set of provided test cases. In addition, during the development of an active learning algorithm, a user has access to the ground-truth models that could be used to test the correctness of the algorithm implementation and might be interested in an oracle that returns all counterexamples between the hypothesis and the SUL. To this end, we provide the "perfect knowledge" equivalence oracle, which always returns the shortest counterexample during deterministic automata learning.

3.7 Auxiliary Functions

To streamline the automata learning experience, we implemented a variety of helper functions. Some of the functions follow naturally the design of AALPY, some were continuously added throughout AALPY's development, and some were identified by the broader automata learning community. Examples of such functions are:

- **Saving, loading, and visualization of models**: AALPY offers the user the ability to save and load models to/from files, as well as to visualize them. AALPY's export syntax follows the .dot format established in the benchmark suite for automata learning [177], and naturally extends it for the model types not present in the community standard.
- **Generation of Random Automata**: AALPY offers the generation of random models for all supported model types. Users can control the model size, the size of the input and output alphabets, decide on the minimality of the model, and in the case of non-deterministic and stochastic models decide on the level of non-determinism.
- **Model Checking**: AALPY provides basic model-checking functionality that enables analysis of stochastic models. This in turn stimulates easier stochastic learning algorithm development, as it provides us with a set of tools with which we can judge the quality of learned models. In particular, AALPY offers the export of MDP and SMM models to PRISM [128] format, as well as the simple statistical model checking module that can be used for bounded reachability analysis [131].

3.8 Empirical Evaluation

In this section, we present an empirical evaluation of algorithms implemented in AALPY. We start by comparing active deterministic learning algorithms with LearnLib. We then briefly compare AALPY's implementation of L^*_{MDP} with the original implementation written in Java [237]. We then evaluate the scalability of our implementations of passive learning algorithms. We do not report on the scalability and the learning performance of non-deterministic and context-free learning, as these learning formalisms are not the central focus of this thesis, and we are yet to apply them in research or practical contexts.

3.8.1 Learning of Deterministic Models.

We compare AALPY's implementation of L^* and *KV* with LearnLib's implementation. We further compare those algorithms with TTT [109]. We compare learning time and the number of learning steps executed during the learning process. Note that we decouple the equivalence query from the learning statistics, that is our evaluation metrics are focused exclusively on total learning time and steps of the learning algorithm itself, and do not consider steps on SUL and time spent in the equivalence query.

Technical Considerations. All experiments were conducted on a Dell Latitude 5410 with an Intel Core i7-10610U processor, and 8 GB of RAM running Windows 10. We compared the latest version of AALPY (1.4.1) with the latest release of LearnLib (0.17.0). We used LearnLib with Java 11 and AALPY with CPython 3.9.

Learning Setup. To enable a fair comparison, we generated all random models with LearnLib and used the same models to evaluate learning on both libraries. Note that from the perspective of the learning algorithms, these models were considered as a black box. Experiments were repeated multiple times to account for the inherent randomness of the equivalence oracles. We attempted to use the same learning and conformance-checking configuration. More concretely, all experiments use the same parameterization of the random walk equivalence oracle, and share all common parameter settings between learning

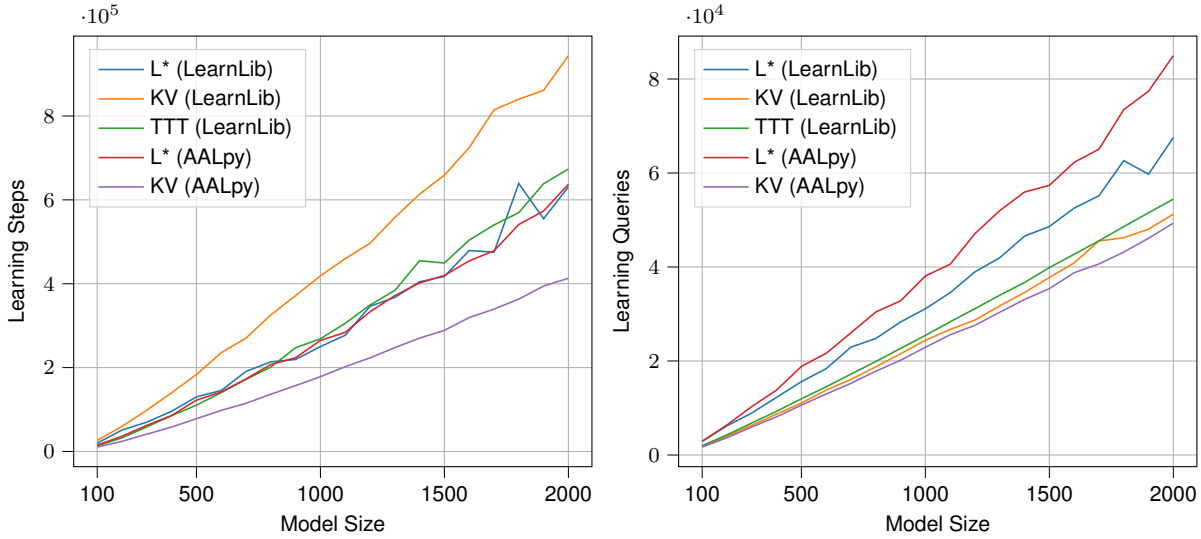


Figure 3.4: Comparison of LearnLib and AALPY on random Mealy machines with an increasing number of states

algorithms. Since we provided an equivalence oracle with sufficient testing resources, learning algorithms always learned models consistent with the models under learning. We used cache in both libraries to minimize the interaction with the SUL, and have used *RS* counterexample processing with all algorithms to optimize the learning process. Code used to compare AALPY and LearnLib can be found at [5].

Comparison on Models of Increasing Size

We start by comparing AALPY's and LearnLib's performance on randomly generated models of increasing size. That is, the randomly generated models have input and output alphabets that have constant size, while the number of states in the model varies. We evaluated the learning algorithms on random Mealy machines with 5 inputs and outputs. The number of states was in the range from 100 to 2000 states, with increments of 100 states. For each state size, we randomly generated 10 different models.

The total number of learning steps (excluding steps asked during the equivalence query) is shown on the left-hand side of Figure 3.4, while the total number of learning queries (input queries asked during the learning) is shown on the right-hand side of Figure 3.4. Results are consistent with observations made in [8]. The minor differences in the number of learning steps between L^* implementations and TTT could be attributed to minor implementation differences and the randomness introduced by the equivalence query, and as expected LearnLib's implementation of *KV* required more interaction with the system than L^* and TTT. However, we observed an interesting outlier: on randomly generated Mealy machines AALPY's implementation of *KV* is even more efficient than LearnLib's TTT implementation. More precisely, AALPY's *KV* implementation required on average $\sim$ 33% fewer learning steps than TTT, and $\sim$ 55% fewer learning steps than LearnLib's implementation of *KV*. However, when we observe the total number of learning queries, shown on the right-hand side of Figure 3.4, we see that both *KV* implementations, as well as TTT, roughly require the same number of learning queries. That implies a substantial difference in average query length, which is a result of the addition of shortened prefixes in the classification tree, as explained in Section 3.4.1.

[5] https://github.com/emuskardin/DeterministicModelLearningComparison

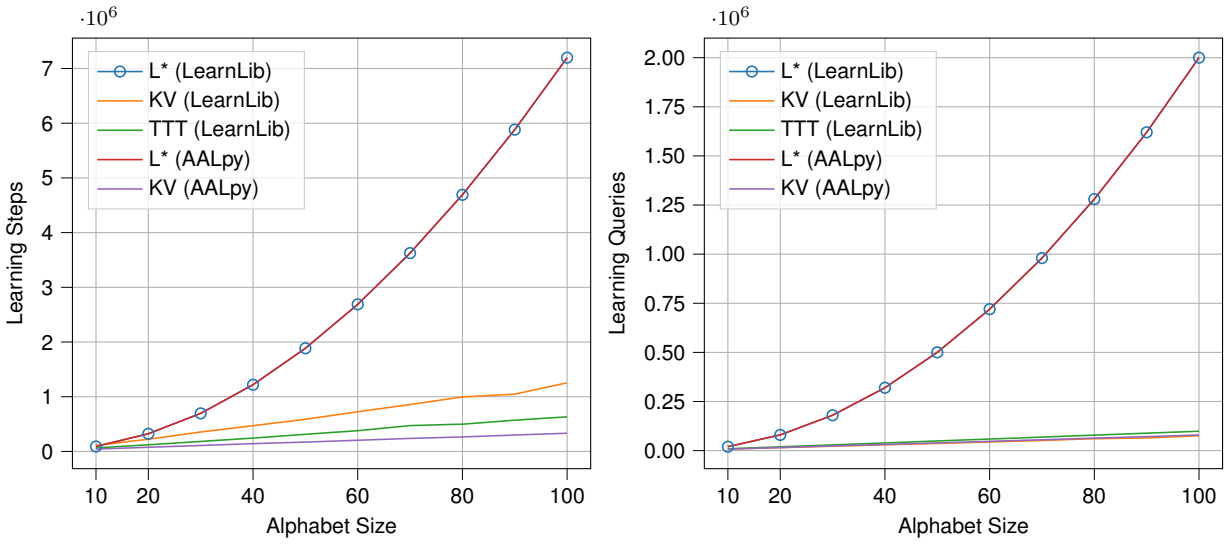


Figure 3.5: Comparison of LearnLib and AALPY on random Mealy machine modes with increasing input alphabet size

Comparison on Increasing Alphabet Size

In this set of experiments, we fix the size of a Mealy machine to 200 states and iteratively increase the size of the input alphabet. We chose to keep the constant output alphabet size of five. The constant size of the output alphabet makes the learning of larger randomly generated models more challenging, as automata learning algorithms have to identify states that are not included in the initial characterization set, which in the case of L^* for Mealy machines consists of the whole input alphabet. The input alphabet was increased in increments of 10, ranging from 10 to 100, and for each input alphabet size we randomly generated 10 Mealy machines.

Figure 3.5 captures the results of these experiments. On the left-hand side of the figure, we observe the number of learning steps required by each algorithm. We observe that the L^* implementation in LearnLib and AALPY have the same number of learning steps for all experiments. This is due to the fact that both implementations learn the complete model in a single learning round, and are therefore not influenced by the randomness introduced by equivalence oracles. *KV* implementations and TTT are substantially more efficient than L^* for this set of experiments. As in the previous set of experiments, AALPY's KV implementation required on average $\sim 33\%$ fewer learning steps than TTT.

Runtime Comparison

We examine runtime for the previous two sets of experiments. Figure 3.6 shows a learning runtime comparison for experiments with increasing model sizes (left) and increasing alphabet size (right).

Let us first examine the runtime on models of increasing size. We observe that LearnLib algorithm implementations are consistently faster than AALPY's implementations. This is not surprising, since the runtime of similarly implemented algorithms favors Java over Python. However, we claim that the runtime differences are negligible since the interaction time with the SUL usually towers over the runtime overhead introduced by the learning algorithm. What is more, total runtime is closely tied to the number of learning steps and queries. This can be best observed in experiments with increasing alphabet sizes. There we observed that both L^* implementations required substantially more learning steps than *KV* and TTT, which then results in substantially higher runtime, as seen on the right-hand side of Figure 3.6.

Note that the runtime difference between LearnLib and AALPY could be minimized by using PyPy[6].

[6] `https://www.pypy.org/`

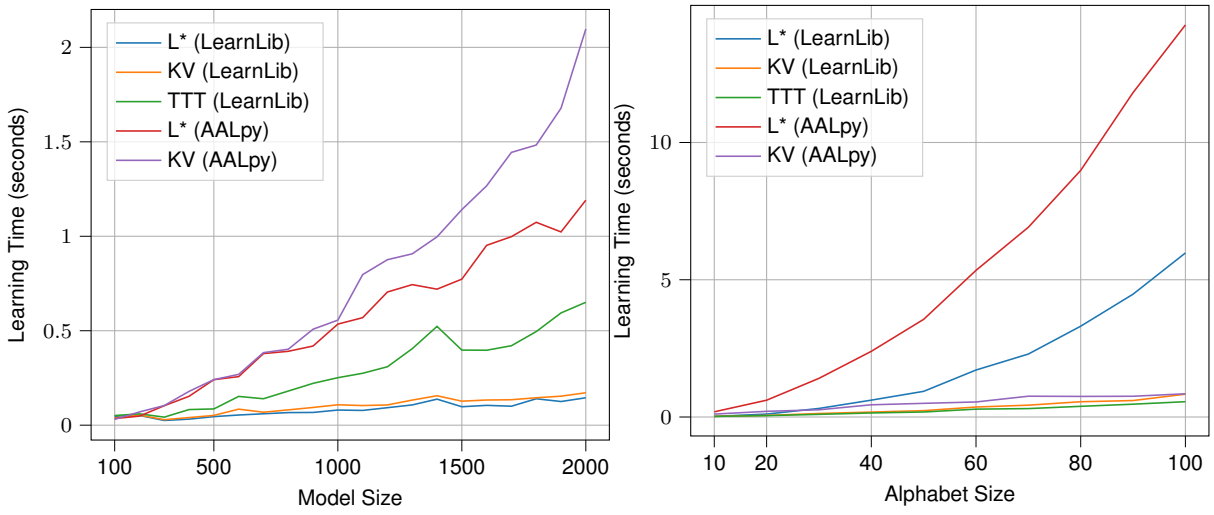


Figure 3.6: Runtime comparison of LearnLib and AALPY on random Mealy machine modes of increasing size (left) and increasing input alphabet size (right)

PyPy is a Python implementation that enables just-in-time compilation, greatly improving the runtime performance over CPython. PyPy could be used as a drop-in replacement for CPython. Even though AALPY is on average 50% faster with PyPy, we do not consider it in the comparison since we assume that the majority of AALPY users will use CPython, and as mentioned previously, we do not consider AALPY's execution time a bottleneck.

Comparison on Selected Benchmark Models

Completely random systems might not be the most suitable benchmark for automata learning algorithm comparison. They can be useful as they can be randomly generated in large quantities and quickly evaluated, and one might assume that a sufficiently large suite of random models will cover all interesting interaction patterns that could be found in models of real systems. However, the generation of such models could still be biased due to the particularities of the used generation algorithm. Therefore, we still conducted a small experiment that considered four models of real-world systems. Such models that capture the input-output behavior of real-world systems often serve as a benchmark for automata learning [177] and are publicly available [7].

We compared AALPY and LearnLib on four models: a model of the TCP Ubuntu client with 15 states, models of VerneMQ and Mosquitto MQTT clients with 17 and 18 states, and an alternating bit protocol (ABP) model with 122 states. The results of this experiment can be seen in Figure 3.7. We selected these models as they are not too "demanding" from the conformance-checking perspective, and we could keep the same consistent equivalence oracle parameterization as done in the previous set of experiments. We learned each model 10 times to account for randomness introduced by an equivalence oracle and observed that both implementations of L^* require more learning steps than both *KV* implementations and TTT. Unlike in previous benchmarks, our implementation of *KV* is slightly less efficient or on par with TTT.

Concluding remark on the comparison with LearnLib. We conclude that both LearnLib and AALPY offer efficient implementations of active learning algorithms that can learn deterministic systems. Since we approach automata learning from a practical perspective, we believe that the choice of an automata

[7] https://automata.cs.ru.nl/

learning library should be selected on a project basis, mostly based on which library and programming language ecosystem enable easier interaction with the SUL.

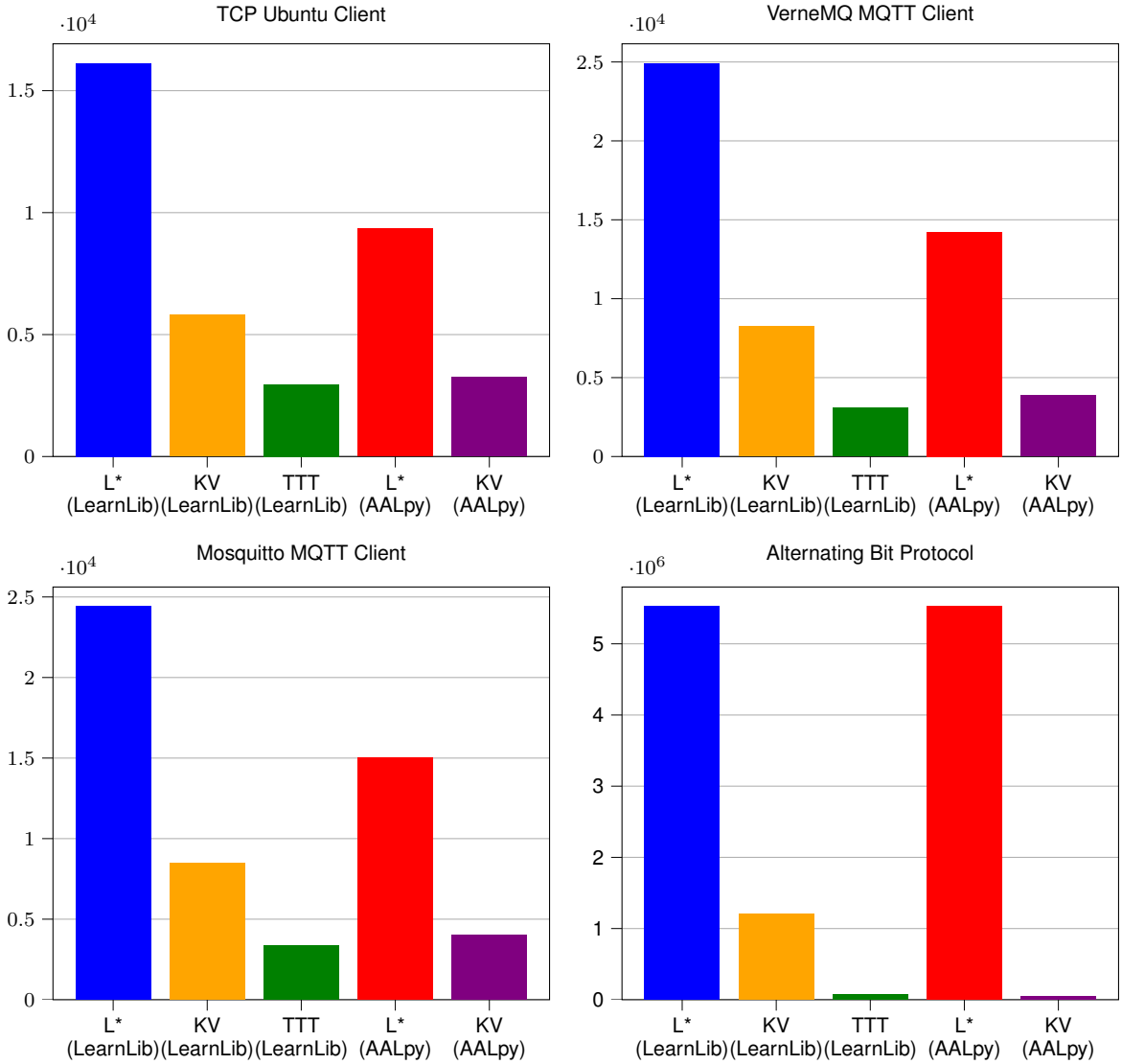


Figure 3.7: Required number of learning steps on simulated models [177] of real-world systems

3.8.2 Active Learning of Stochastic Models

We evaluated AALPY on learning stochastic models with the same experiments as the original Java version of L^*_{MDP} [237]. That is, we learned MDPs by simulating known ground-truth MDP models as black boxes and measured the learning runtime and accuracy. To measure accuracy, we used a probabilistic model-checker to compute probabilities for satisfying temporal properties with the ground-truth models and the learned models. The model-checking error then quantifies accuracy, which we compute as the absolute difference between the results on the ground truth and the results on the learned models. Figure 3.8 shows the average runtime and the average model-checking errors measured in the experiments. We can see that the AALPY and the Java implementation are generally similarly fast and produce similarly accurate models. Evaluation differences can be attributed to minor implementation details.

In our experiments both implementations used random word oracles. However, in the original paper [237] the learning algorithm relied on the scheduler-based equivalence oracle. This oracle introduced significant computational overhead, which resulted in substantially slower learning while providing sim-

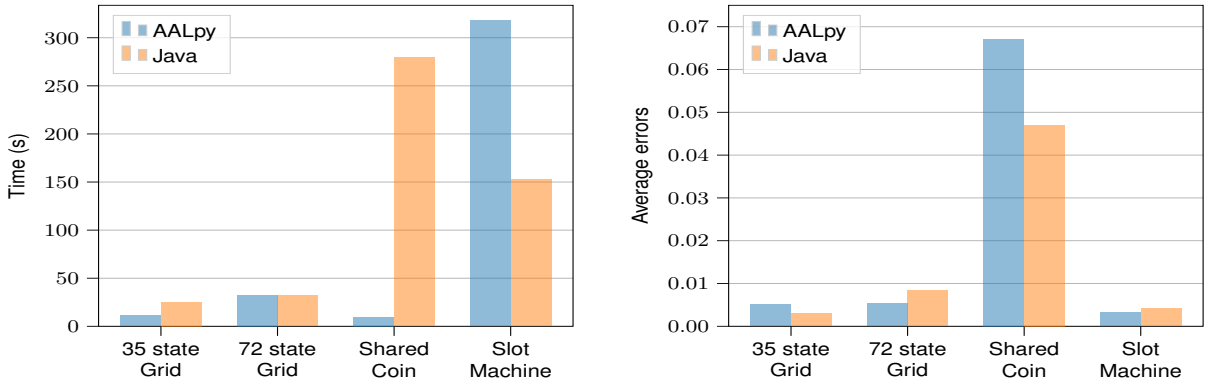


Figure 3.8: Runtime measurements and probabilistic model-checking errors on learned models for the AALPY implementation and the Java implementation of L^*_{MDP}

ilar learning results as random oracles. This finding provides validity to the assumption that random testing sufficiently explores the SUL state space in non-deterministic formalisms.

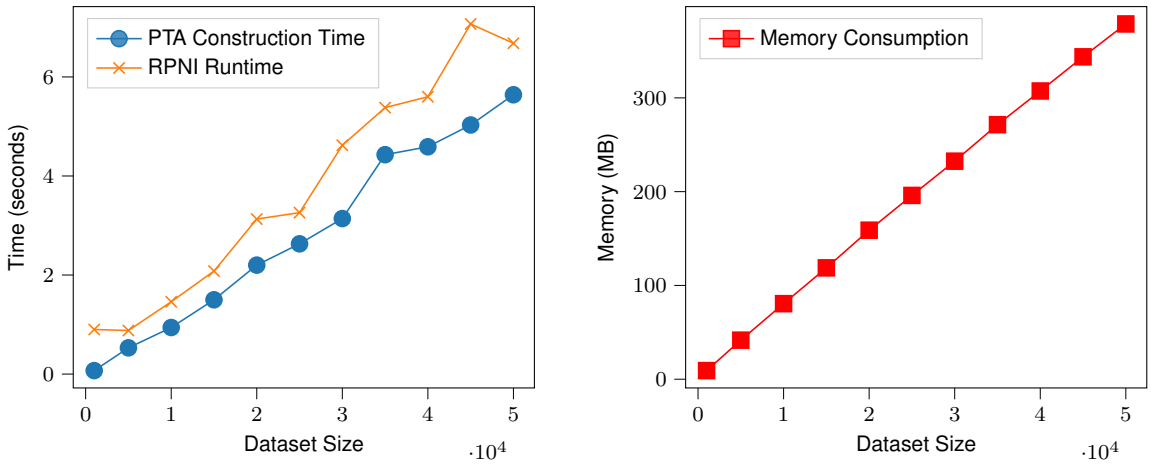


Figure 3.9: Influence of dataset size on the RPNI's runtime and memory consumption

3.8.3 Scalability of Passive Learning Algorithms

In this section, we briefly examine the scalability of our passive learning algorithm implementations. More precisely, we examine their memory footprint and runtime with respect to the number of provided sequences.

Setup. We generated a random Mealy machine and random MDP with 50 states and 5 elements of input and output alphabet. The former was used for the RPNI evaluation, while the latter was used with ALERGIA. For each model, we evaluated runtime performance and memory consumption with an increasing size of the dataset. Dataset sizes were in the range of 1000-50000 traces with increments of 5000. All sequences consist of uniformly sampled inputs and are of length in range [5-30]. We report on the runtime and memory consumption of CPython and have observed that PyPy is in general 1.5-3 times faster on these benchmarks.

Results. Figures 3.9 and 3.10 show the runtime and memory consumption for passive learning of deterministic and stochastic systems. We split the (F)PTA construction time and algorithm execution time. The total runtime of the passive learning process is the sum of these values. We observe that runtime

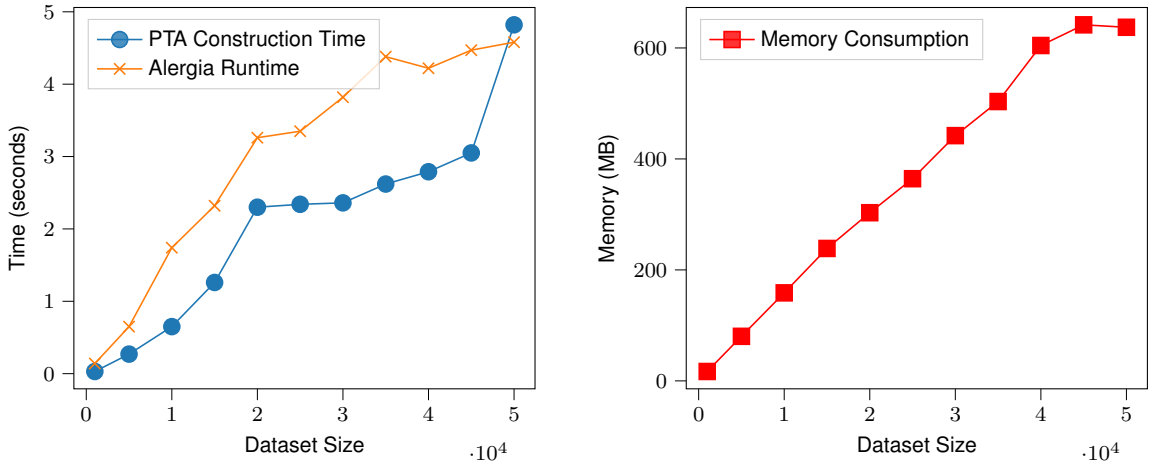


Figure 3.10: Influence of dataset size on the ALERGIA's runtime and memory consumption

and memory consumption increases linearly for both RPNI and ALERGIA. However, from the figures we can observe that ALERGIA has more variability, and this is due to the MDP's inherent randomness that influences the size of the FPTA even with the same input sequences. We believe that both algorithms are sufficiently efficient, both in runtime and memory consumption, and can be used in many practical applications.

3.9 Selected Applications

In subsequent chapters of the thesis, we present several applications of AALPY in the machine learning domain. In this section, we outline other domains and specific use cases that relied on AALPY.

Finding bugs in VIM. AALPY has been used as a debugging tool for software that is internally based on a state machine, more specifically for the text editor Vim and its feature-enriched fork Neovim. A group of researchers used AALPY to learn deterministic models of VIM modes, and during the learning process, non-deterministic behavior was observed. Since they were learning deterministic models, AALPY terminated the learning algorithm and returned the input sequences that display non-deterministic behavior. After the examination of non-deterministic sequences, they were able to isolate the root causes and submit bug reports. At least five bugs were found, reported, and later fixed by the VIM community[8]. A more detailed description of this case study was presented at SPIN 2024 [78].

Part for Lcap Library. Some of AALPY's functionalities have been used as a foundation of the `Lcap` learning library[9]. `Lcap` provides learning algorithms that can learn language intersections [114]. To this end, they followed the three-step active learning process described in Section 3.4: they implemented custom `SUL` classes that enable active learning of language intersections, implemented custom oracles largely based on the Random-W method for equivalence checking and have used L^* for the learning itself.

Learning and Testing of Bluetooth Low-Energy protocol. Pferscher and Aichernig [194, 196] used AALPY to learn the connection interface of BLE devices. Using a learning library implemented in Python creates the opportunity for a smooth integration of communication package libraries like SCAPY [217]. In this application, SCAPY was used to construct BLE packages also on lower levels of the BLE protocol

[8] https://github.com/DES-Lab/AALpy/discussions/13
[9] https://github.com/sjunges/Lcap

stack. Furthermore, the case study on the BLE protocol shows that AALPY can be extended with a fault-tolerant SUL implementation. Fault tolerance is especially necessary for the learning of communication protocols since requests or responses might be delayed or lost. Additionally, AALPY's caching mechanism reduces the costs of time-expensive network communication. In their case study, they learned the behavioral models of five BLE devices. The learned models were different for every device. Considering the differences in the behavioral models, a fingerprinting sequence could be generated that uniquely identifies the BLE device. Their learning framework was later used to enable learning-based fuzzing of BLE devices [197].

Learning of Git version control system. AALPY has been used to learn behavioral models of the Git version control system [170]. More specifically, Git interactions were performed through the command line interface, as well as through GitPython, the popular Python Git library. Learned models from both Git interfaces were compared and found differences revealed inconsistencies between implementations. These inconsistencies were either caused by intentional design decisions made by the GitPython development team or displayed unintended behavior, that is bugs.

3.10 AALPY's Position in Automata Learning Landscape

LearnLib [111]. Before we developed AALPY, we were users of LearnLib. Our experiences with LearnLib shaped many design decisions made during AALPY's development. LearnLib and AALPY implement many common algorithms and functionalities, but differ in their focus: LearnLib goes more in-depth, while AALPY's functionality extends in the breadth. This view, which is a generalization, was used throughout the development: AALPY should not compete with LearnLib in its offer of a variety of different algorithms for deterministic model learning, but should extend LearnLib capabilities to non-deterministic and stochastic learning. This can be seen in the availability of implemented algorithms: at the time of the writing of the thesis, LearnLib offers nine active deterministic learning algorithms and two passive deterministic learning algorithms, while AALPY offers only two deterministic active and one deterministic passive learning algorithms, but also offers active and passive stochastic learning algorithms. Both libraries also implement, albeit different, non-deterministic and context-free learning algorithms. Another difference between AALPY and LearnLib lies in their implementation. LearnLib provides a modular design that enables easy modification of existing algorithms, while AALPY algorithm implementations are, compared to Learnlib's, less extensible. Other implementation differences can be attributed to differences in used programming languages, design principles, and development practices. As concluded in Section 3.8.1, we believe that the choice of used automata learning library should be made on a project basis. LearnLib could be more attractive for developers of active automata learning algorithms for deterministic models, as it offers a wider selection of said algorithms that are available for comparison. On the other hand, AALPY offers algorithms that can learn stochastic models and generally enables simpler application of automata learning in various domains, due to Python's popularity as an interface language.

LibAlf [31] LibAlf provides an open-source implementation of several active and passive learning algorithms. It is written in C++ under the LGPLv3 license. LibAlf appears to be in its final state, as the last commits to its repository were made four years ago. It features implementations of L^* and *KV* capable of learning DFAs and Mealy machines, as well as an implementation of RPNI. LearnLib offers a more mature automata learning framework and provides more efficient implementations of learning algorithms compared to LibAlf [111].

FlexFringe [255]. *flexfringe* is a passive automata learning tool that provides a generalized passive automata learning framework. *flexfringe* is largely based on a slight adaptation of the red-blue framework

presented in Section 2.6. Its modular design allows users to adapt the red-blue framework for a variety of learning formalisms, including deterministic, non-deterministic, and stochastic passive learning. The generalized framework enables efficient implementations of RPNI (and its evidence-driven variants [130]) and Alergia. AALPY's implementation of the generalized state merging framework shares many similarities with *flexfringe*.

JajaPy [213]. JAJAPY is the newest addition to the existing automata learning tool landscape. JAJAPY focuses on stochastic passive learning of various types of Markov models. It features the Baum-Welch algorithm for learning of Markov models, as well as an implementation of ALERGIA. The JAJAPY also offers an active variant of implemented passive algorithms. JAJAPY authors evaluated their algorithms against AALPY, and have concluded that AALPY and JAJAPY complement each other in the domain of stochastic model learning, and the appropriate library should be chosen based on the specific use case.

3.11 Concluding Remarks

In this section, we have presented AALPY, an automata learning library written in Python. AALPY features efficient implementations of various automata learning algorithms that can be used to learn deterministic, non-deterministic, and stochastic models, as well as context-free languages. We hope that AALPY will contribute to the wider adoption of automata learning, and that it will continue to be used by educators, researchers, and industry alike.

We plan to continue maintaining and continuously improving AALPY for the foreseeable future. We also plan to further extend AALPY, both with additional learning algorithms, such as resetless learning [89], as well as with other modeling formalism, such as register automata [110].

RQ 1. How can automata learning techniques be seamlessly employed in the machine learning domain?

The design of AALPY enables efficient applications of state-of-the-art automata learning algorithms. We provided a simple interface to existing learning algorithms that can learn deterministic, non-deterministic, stochastic, and context-free models. Through this interface, users can interchangeably use all available algorithms. In this chapter, we have outlined AALPY's design and functionalities, and we point out that all subsequent chapters of this thesis further strengthen our claims about AALPY's versatility, especially in relation to **RQ 1**. This point is also strengthened by related work that relied on AALpy [10, 35, 114, 165, 196, 197, 198]. Finally, AALPY's ease of use was even pointed out by the developers of other automata learning libraries [212, 256].

RQ 1.1 How can existing automata learning approaches be improved to enable efficient automata learning?

We outlined several heuristics that improve upon existing automata learning state-of-the-art. Examples of such heuristics are modified counterexamples processing in *KV*, generalization of RPNI to Moore machine learning, relaxation of an all-weather assumption for non-deterministic learning, a memory-optimized variant of ALERGIA that ensured higher accuracy of the learned model, among others. Such improvements were continuously added to AALPY since its initial release, and we claim that AALPY's simple structure enables quick prototyping, testing, and inclusion of such heuristics. We further present a substantial algorithmic improvement in learning of stochastic models in the next chapter.

4

ACTIVE MODEL LEARNING OF STOCHASTIC REACTIVE SYSTEMS

Declaration of Sources

The chapter is based on the conference and journal publications "Active Model Learning of Stochastic Reactive Systems" [171, 238]. A conference version of this work was presented at SEFM2021 [238], where it won the best paper award. We were invited to present the extension of our work in the Software and Systems Modeling (SoSyM) journal [171], and the extended version was accepted in February 2024.

4.1 Motivation

The majority of research in the automata-learning domain considers algorithms that learn models of deterministic systems [14, 109, 111, 116]. However, systems that rely on randomized distributed algorithms [19] are often modeled with MDPs and analyzed with techniques such as probabilistic model checking [127]. In addition, as outlined in Section 2.6 the majority of RL problems are formalized on MDPs, that is, we assume that the RL environment's structure could be modeled as an MDP and the goal of an RL agent is to compute an optimal policy that solves some task. To tackle such challenges, researchers have proposed stochastic model learning methods [41, 144, 235, 237], upon which we build upon and extend in this chapter.

Let us first start by examining the differences between deterministic, non-deterministic and stochastic models. In a deterministic setting executing an input from a state always produces the same output, while in a non-deterministic and stochastic setting, a single input might produce multiple outputs. Stochastic systems produce outputs that are distributed probabilistically. Stochastic learning algorithms accordingly learn models where output distributions are approximated based on the data observed during learning. In the non-deterministic setting, outputs are not sampled from probability distributions but arise due to the inherent non-deterministic behavior of the system. In practice, non-determinism often arises from abstraction [2], more precisely, non-determinism is caused by imprecisions introduced by such abstraction. Non-deterministic behavior can also indicate the presence of unintended behavior or bugs in deterministic systems, as observed in [234].

The assumption that the SUL behaves deterministically greatly simplifies the automata learning process, since according to this assumption an input sequence always results in the same output sequence.

This makes obtaining new information about the SUL trivial since once we have identified parts of the SUL that need to be further explored, we construct an input query that returns the desired information. In contrast, the probabilistic nature of stochastic systems complicates the information-gathering process, since executing an input query could result in many different output sequences. As such, an interesting part of the SUL might not be easily reachable, since the inherent randomness of the SUL might steer our input query in another direction. And since every input could produce multiple different outputs according to some probability distribution, we need to develop a state-comparison method that accounts for the probabilistic nature of the system and for the potential lack of information.

Tappler et at. [235, 237] were the first to tackle these challenges in the active learning setting, and they presented an L^*-based learning algorithm that allows one to actively learn MDP models that capture the input-output behavior of stochastic systems. In this chapter, we further improve upon their work by introducing L^*_{SMM}, an L^*-based approach for learning stochastic Mealy machines (SMMs), along with algorithmic improvements that also apply to the learning of MDPs.

4.1.1 Guiding Observations

During the development of AALPY, we made two observations that served as the initial guiding concepts on which we developed some of the content presented in this chapter.

First, the more obvious observation is that the deterministic active automata learning of an SUL is more efficient if the SUL's input-output behavior is encoded as a Mealy machine rather than as a Moore machine. This is mainly a consequence of the fact that the same input-output behavior could be captured by a smaller Mealy machine compared to an observationally equivalent Moore machine. Since Mealy machines could have fewer states than equivalent Moore machines, the active learning algorithms need to maintain a smaller observation table and ask fewer queries in order to construct a hypothesis. Remember that the complexity of L^* is polynomial with respect to the number of states of the final model and the size of the input alphabet, and since a Mealy machine could have fewer states than an equivalent Moore machine, learning of Mealy machines might lead to more efficient learning.

Another important observation that greatly improved the stochastic model learning process relates to the stopping criterion. We show how the stopping criterion presented in [235, 237] could be adapted in a way that will terminate the learning process as soon as the "point of diminishing returns" is identified. As we will explain later in this chapter, the termination of stochastic learning algorithms is not necessarily as straightforward as for deterministic learning, and terminating the learning process at the point after which the accuracy of the learned model does not substantially increase is paramount for efficient learning.

Both of these observations were made during the initial development of AALPY, and during this initial development, we further realized a need for a simplified active stochastic learning algorithm. Therefore, the presented version of the algorithm is less dependent on the user's parameterization, while maintaining all formal properties and even improving the performance.

Structure. This chapter is structured as follows. In Section 4.2, we introduce the stochastic Mealy machines formalism. In Section 4.3 we present the learning algorithm. Firstly we present the theoretical formalisation of the most important aspects of the learning process, and then we outline the implementation details. We then evaluate in Section 4.4, and then we show how IOALERGIA could trivially be adapted to learn SMMs. Finally, in Section 4.6 we present concluding remarks and position the presented work with respect to the research questions.

4.2 Stochastic Mealy Machines

As the name implies, stochastic Mealy machines are a probabilistic adaptation of deterministic Mealy machines. Stochastic Mealy machines are defined as follows:

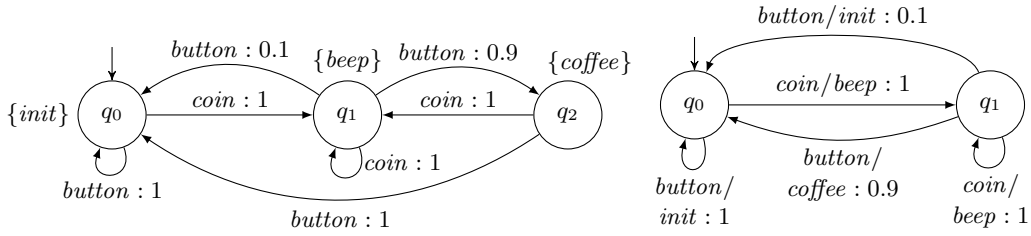


Figure 4.1: An MDP model (left) and an SMM model (right) of a faulty coffee machine

Definition 4.1 (Stochastic Mealy Machines (SMM)).
A stochastic Mealy machine is a 5-tuple $\langle Q, I, O, \delta, q_0 \rangle$, where

- *Q is a finite set of states,*
- *I is a finite set of input symbols (input alphabet),*
- *O is a finite set of output symbols (output alphabet),*
- *$\delta : Q \times I \to Dist(Q \times O)$ is the probabilistic transition function,*
  - *where Dist(X) is the set of all probability distributions over X, and*
- *$q_0 \in Q$ is the initial state.*

Analogously to Mealy and Moore machines, SMMs and MDPs differ in the way outputs are produced. An SMM produces an output considering its current state and input, whereas the output of an MDP depends only on the current state. Like in a deterministic setting, SMMs are potentially smaller, as illustrated in Figure 4.1.

On the left-hand side of Figure 4.1, we show an MDP modeling a faulty coffee machine, and a corresponding SMM on the right-hand side. Both models are observationally equivalent, except for the initial output produced by the MDP. We show inputs and probabilities as edge labels, s.t. the probabilities follow after a colon. In the MDP, the outputs are defined by the labels on the states, whereas for the SMM they are part of the edge label, separated from the input via a slash.

Note that MDPs and SMMs differ only in how the output is produced, and as a consequence of the different output functions, SMMs do not have an explicit output tied to an initial state. Therefore, we slightly adapt the notion of SMM traces compared to MDPs.

We use $q \xrightarrow{i \cdot o} q'$ to denote $\delta(q, i)(q', o) > 0$ and extend this notation to traces t in $(I \cdot O)^*$ by $q \xrightarrow{\epsilon} q$ and $q \xrightarrow{i \cdot o \cdot t} q'$ if $\exists q'' : q \xrightarrow{i \cdot o} q'' \wedge q'' \xrightarrow{t} q'$. We consider input-enabled systems, such that δ is total and thus defined for all $q \in Q$ and $i \in I$. Furthermore, we consider *deterministic* SMMs, such that for every trace t, there is exactly one path producing t. Consequently, while an input may cause different outputs, an input-output pair cannot lead to different states. This property is analogous to the one formalized on MDPs in Section 2.1.2.

The formalization further enables a standard transformation from SMMs to MDPs, similarly as between deterministic Mealy and Moore machines. To transform an SMM into an MDP, we create an MDP state for every pair of SMM state s and output of an incoming transition to s. Additionally, we create a new state with a special unused output label that serves as the initial MDP state. Then, we create transitions between all states corresponding to the transitions in the SMM.

Finally, we define the trace equivalence as an adaptation of the Nerode relation for regular languages [179], and equivalence of SMMs. Thus, let us define input-output semantics $[\![\mathcal{M}]\!]$ that maps traces followed by an input to the output distribution produced by the input. Formally, $[\![\mathcal{M}]\!] : (I \cdot O)^* \cdot I \to Dist(O) \cup \{\bot\}$ with $[\![\mathcal{M}]\!](t \cdot i) = \lambda(q, i)$ if there is a $q \in Q$ such that $q_0 \xrightarrow{t} q$ and $[\![\mathcal{M}]\!](t \cdot i) = \bot$ otherwise.

Definition 4.2 (Trace Equivalence).
Two traces $t, t' \in (I \cdot O)^*$ are equivalent if for all continuation sequences $cs \in I \cdot (O \cdot I)^*$:

$$[\![\mathcal{M}]\!](t \cdot cs) = [\![\mathcal{M}]\!](t' \cdot cs).$$

Table 4.1: Observation table for the faulty coffee machine shown in Figure 4.1.

		button	*coin*
S	ϵ	$\{$*init*: 247$\}$	$\{$*beep*: 414$\}$
	coin · *beep*	$\{$*coffee*: 147, *init*: 16$\}$	$\{$*beep*: 134$\}$
$S \cdot I$	*button* · *init*	$\{$*init*: 69$\}$	$\{$*beep*: 82$\}$
	coin · *beep* · *button* · *coffee*	$\{$*init*: 64$\}$	$\{$*beep*: 53$\}$
	coin · *beep* · *button* · *init*	$\{$*init*: 9$\}$	$\{$*beep*: 6$\}$
	coin · *beep* · *coin* · *beep*	$\{$*coffee*: 65, *init*: 7$\}$	$\{$*beep*: 61$\}$

Definition 4.3 (SMM Equivalence).
Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two SMMs over the same input and output alphabets. $\mathcal{M}_1$ and $\mathcal{M}_2$ are equivalent iff for all test traces:

$$ts \in (I \cdot O)^* \cdot I : [\![\mathcal{M}_1]\!](ts) = [\![\mathcal{M}_2]\!](ts).$$

Two traces are equivalent if they produce the same output distributions in response to the last input of every continuation sequence. Likewise, two SMMs are equivalent if they produce the same output distributions for all test sequences. However, since the learner is not aware of the exact output distribution produced by the SUL, but rather only observes output frequencies, we use statistical tests to approximate the equivalence of output distributions.

4.3 Active Learning of Stochastic Mealy Machines

In this section, we present L^*_{SMM}, an algorithm for learning SMMs. For this purpose, we extend, adapt, and improve our L^*-based algorithm for MDPs [235, 237]. First, we define the setting and basics for learning SMMs, such as semantics. Next, we introduce the queries that are used in the interaction between learner and teacher. After that, we discuss learning itself and present implementation details on selected aspects, such as queries, counterexample processing, and stopping. We conclude the section with a complexity analysis and a discussion of convergence.

Let us start by examining an example of an observation table that is maintained by L^*_{SMM}. An example of such an observation table can be seen in Table 4.1. We use this observation table to highlight certain challenges that arise during active model learning of stochastic systems, and through this chapter, we present how these challenges are handled. We immediately observe that cells of the observation table contain output frequency values. That is, for a cell defined with trace t that contains a prefix from the S set and a suffix from the E set, we do not store a single output value as in deterministic learning, but all outputs and their observed frequencies that were observed executing trace t multiple times on the SUL. However, this complicates the row compatibility which is used during closing and hypothesis creation. In deterministic learning, two rows in the observation table are compatible if all of their cell values are equivalent, whereas in stochastic learning two rows are deemed equivalent if all of their cells pass the statistical compatibility check. Therefore we start the presentation of the learning algorithm by introducing statistical compatibility tests to determine cell, and consequently cell equivalence.

4.3.1 Statistical Tests to Determine Trace Equivalence

Since we sample traces with outputs distributed according to SUL's semantics, we cannot determine the exact equivalence of output distributions as required by Def. 4.2. We instead perform statistical tests for differences between sampled output frequencies.

To formalize the problem of approximating equivalence checking between sampled frequencies, let tc_1 and tc_2 be two sequences in $(I \cdot O)^* \cdot I$ and let $\mathcal{T}$ be a multiset of traces collected from the SUL $\mathcal{M}$. We define the frequency function as $\mathbf{freq}_{\mathcal{T}}(tc) = o \mapsto \mathcal{T}(tc \cdot o)$ for $o \in O$, extend the notion

of support *supp*() to frequencies, and introduce *Freq*(O) for $O \to \mathbb{N}_0$, the set of all output frequency functions. Our goal is to approximate $[\![\mathcal{M}]\!](tc_1) \neq [\![\mathcal{M}]\!](tc_2)$ by testing whether $f_1 = \mathbf{freq}_{\mathcal{T}}(tc_1)$ and $f_2 = \mathbf{freq}_{\mathcal{T}}(tc_2)$ have been sampled from different distributions. We say that the frequencies f_1 and f_2 are compatible, denoted $f_1 \approx f_2$, if they are not different. Let $n_i = \sum_{o \in O} f_i(o)$. In the special case that $n_1 = 0$ or $n_2 = 0$, we define $f_1 \approx f_2$ to hold, since then we do not have sufficient information to detect a difference.

We implemented two difference checking strategies: (1) one based on Hoeffding bounds [96], which are also used in other stochastic automata learning algorithms [41, 237], and one based on Pearson's χ^2 test [76] for testing homogeneity of multinomial distributions.

Both strategies come with advantages and drawbacks. Using Hoeffding bounds, we view the occurrence of each output as a Bernoulli-distributed random event. This means we must perform several individual tests for outputs produced in response to a single state-input pair. Hence, errors may compound. The application of Hoeffding bounds is motivated by its application in IOALERGIA [144] and its predecessor ALERGIA [41]. In contrast to ALERGIA which learns probabilistic finite automata, we learn MDPs, where outputs follow a categorical distribution rather than a Bernoulli distribution. Hence, Pearson's χ^2 test may be a better fit. However, this test places stricter requirements on the amount of sampled data. Especially "almost deterministic" systems and systems with rare outputs, in general, may be problematic, as the χ^2 distribution may not fit well in such cases. Due to the relevance of such domain-dependent considerations, we will empirically evaluate both strategies on various systems in Section4.4.

Hoeffding check. For $n_1 > 0$ and $n_2 > 0$, $\alpha \in (0, 2]$ defining the significance level. Frequencies f_1 and f_2 are different if $\exists o \in O$:

$$\left| \frac{f_1(o)}{n_1} - \frac{f_2(o)}{n_2} \right| > \sqrt{\frac{1}{2} \ln \frac{2}{\alpha}} \left(\frac{1}{\sqrt{n_1}} + \frac{1}{\sqrt{n_2}} \right). \tag{4.1}$$

(correct with probability $\geq (1 - \alpha)^2$ [41])

Chi-squared (χ^2). Let $O_1 = \mathit{supp}(f_1)$ and $O_2 = \mathit{supp}(f_2)$. The frequencies f_1 and f_2 are different if

$$\sum_{i=1}^{2} \sum_{o \in O_1 \cup O_2} \frac{f_1(o) - n_1 \cdot \hat{p}_o}{n_1 \cdot \hat{p}_o} \geq \chi^2_{1-\alpha, |O_1 \cup O_2| - 1} \tag{4.2}$$

$$\text{where } \hat{p}_o = \frac{f_1(o) + f_2(o)}{n_1 + n_2}. \tag{4.3}$$

There are two special cases affecting the applicability of the χ^2 test, which may occur for long sequences tc with few observations. If $O_1 \cap O_2 = \emptyset$, we apply the Hoeffding test defined above as a fallback. If $|O_1 \cup O_2| = 2$ and there is an $o \in O_1 \cup O_2$ with $f_i(o) \leq 5$, we apply the χ^2 test with the Yates correction [277].

The two different tests lead to varying learning behavior and performance. For example, if we expect mostly non-stochastic behavior from the SUL, the requirements for the standard χ^2 test may often be violated. In such cases, we apply the Yates correction, which may result in more pessimistic checks, thus requiring more samples to detect a difference. On one hand, this leads to worse performance, but on the other hand, it leads to very accurate learned models according to our experiments. The worse performance of pessimistic checks with the Yates correction stems from the fact that fewer difference verdicts mean that the intermediate hypotheses have a lower number of states as more observation table rows would be compatible; see hypothesis construction below. As a result, learning might take more sampling to converge. The final models, however, may be more accurate than with the Hoeffding check simply because more data is available to estimate output distributions. Hence, the choice of difference check depends on the application domain and the time budget for learning.

4.3.2 Query Types

As with other active learning algorithms, we formalized L^*_{SMM} in the MAT framework. The learner and teacher interact with each other via two types of queries: (1) *tree queries* and (2) *equivalence queries*. The teacher samples traces for both query types but with different goals. Tree queries attempt to gain more accurate information about the SUL, whereas equivalence queries attempt to falsify a hypothesis SMM – as proving equivalence is not possible in a black-box setting, we target falsification. Note that a tree query is a generalization of a standard input query found in the MAT framework.

- Tree query ($\mathbf{tq}$): Let *FT* be an IFOPT (as defined in Section 2.3.2), a tree query $\mathbf{tq}(FT)$ returns a multiset of traces from $(I \cdot O)^*$, where inputs are chosen according to *FT* and outputs are sampled according to the SUL semantics.
- Equivalence query ($\mathbf{eq}$): Let $\mathcal{H}$ be a hypothesis SMM, an equivalence query $\mathbf{eq}(\mathcal{H})$ returns a pair $(r, \mathcal{T}_{\text{cex}})$ where r is the query result in $\{yes\} \cup (I \cdot O)^* \cdot I$ and $\mathcal{T}_{\text{cex}}$ is a multiset of traces sampled for the query.

For a tree query $\mathbf{tq}$, the learner creates an IFOPT *FT* and asks the teacher to sample paths from *FT*, while selecting inputs i with a probability proportional to the frequency value of i in *FT*. The result of a $\mathbf{tq}$ is a multiset of sampled SUL traces. For an equivalence query, the learner forms a hypothesis SMM $\mathcal{H}$ and asks if the hypothesis is equivalent to the SUL. The teacher responds either with *yes* or with a counterexample to equivalence in $(I \cdot O)^* \cdot I$. Additionally, the teacher returns a multiset of traces that have been sampled to perform the query.

4.3.3 Learner

Data Structures

The learner uses two main data structures, a multiset $\mathcal{T}$ of sampled traces and an observation table, a triple $\langle S, E, T \rangle$. Function $\mathbf{freq}_{\mathcal{T}}(tc)$, returns the observed output frequencies, while S, E, and T are defined by:

- $S \subseteq (I \cdot O)^*$ is a prefix-closed set of traces,
- $E \subseteq (I \cdot O)^* \cdot I$ with $I \subseteq E$ is a suffix-closed set of continuations, and
- $T\colon (S \cup S \cdot I) \cdot E \to \mathit{Freq}(O)$ with $S \cdot I = \{s \cdot i \cdot o \mid s \in S, i \in I, o \in O\colon \mathbf{freq}_{\mathcal{T}}(s \cdot i)(o) > 0\}$ and $T(s \cdot e) = \mathbf{freq}_{\mathcal{T}}(s \cdot e)$ stores output frequencies.

An observation table can be represented as a two-dimensional table with rows labeled by *short* traces in S and by *long* traces in $S \cdot I$, columns labeled by continuation sequences in E, and with cell content given by T. Table 4.1 shows an observation table from a learning run of L^*_{SMM} on the coffee machine shown in Figure 4.1.

As is commonly done in L^*-based learning, we create hypotheses as follows. We partition S based on the row content given by functions $row(s) : E \to \mathit{Freq}(O)$ with $row(s)(e) = T(s \cdot e)$ and create a state for every block in the partition. We further create transitions for input-output pairs $i \cdot o$ between blocks b and b' by determining the block b containing an $s \in S$ and the block b' containing its input-output extension $s \cdot i \cdot o$, if it exists. The long traces $S \cdot I$ ensure that traces can be created for all observed input-output pairs. To partition S, we extend the notion of frequency compatibility to rows and adapt a technique introduced in previous work [237], as compatibility is not an equivalence relation. We say that two rows labeled by traces s and s' are compatible if all their cells are compatible, i.e., $\forall e \in E : row(s)(e) \approx row(s')(e)$. We also say that the traces s and s' are compatible. In Table 4.1, the first, third, fourth, and fifth row contain different frequencies, but by normalizing we can see that they

Algorithm 4.1 Creation of compatibility classes

for all $s \in S$ **do**
  $\text{rank}(s) \leftarrow \sum_{i \in I} \sum_{o \in O} \widehat{T}(s \cdot i)(o)$
unpartitioned $\leftarrow S$
$R \leftarrow \emptyset$
while *unpartitioned* $\neq \emptyset$ **do**
  $r \leftarrow m$ where $m \in$ *unpartitioned* with largest $\text{rank}(m)$
  $R \leftarrow R \cup \{r\}$
  $cg(r) \leftarrow \{s \in$ *unpartitioned* $\wedge\ \mathbf{compatible}_E(s, r)\}$
  for all $s \in cg(r)$ **do**
    $rep(s) \leftarrow r$
  unpartitioned $\leftarrow$ *unpartitioned* $\setminus\ cg(r)$

model exactly the same (empirical) probability distributions. Hence, they would be pairwise compatible and thus be in the same block of a partition of S and correspond to the same state in the hypothesis derived from Table 4.1. The situation for the second and last row is a bit different. Their respective first cells contain the same outputs, but the empirical probability distributions estimated from them are different. However, the difference is small enough that we would deem the cells, and thus the rows, compatible according to both statistical tests. As an example where the tests disagree, suppose that the cells contain $\{\textit{coffee}\colon 147, \textit{init}\colon 7\}$ and $\{\textit{coffee}\colon 65, \textit{init}\colon 16\}$, respectively, i.e., frequencies values would be swapped. Using an $\alpha = 0.05$ for both, the Hoeffding check would deem the cells compatible, whereas the χ^2-squared tests would deem them incompatible. The reason is that the former looks at outputs individually, whereas the latter looks at the whole cell at once, thus noticing a difference. Hence, the Hoeffding check would be more aggressive, differentiating fewer rows and thus creating a model with fewer states.

We face the difficulty that compatibility is not an equivalence relation, as transitivity does not hold in general. A row may be compatible with multiple other rows that are not necessarily pairwise compatible. To tackle this challenge, we create compatibility classes $cg(r)$ that partition S, like in our previous work [237]. Each compatibility class $cg(r) \subseteq S$ has a unique representative r in the set of representatives $R \subseteq S$ and every trace t in $cg(r)$ is compatible to r. The function $rep(t) = r$ returns the unique representative r for short and long traces t. We create compatibility classes by iteratively selecting a new representative r from the unpartitioned part of S and greedily adding other unpartitioned traces to $cg(r)$. The selection of a new representative r is based on how often r was observed during sampling, which ensures that $\epsilon \in R$ can be used as the initial hypothesis state. Algorithm 4.1 shows the compatibility classes computation process.

Hypothesis Generation

To create a hypothesis from an observation table $\langle S, E, T \rangle$, the table must be closed and consistent. Adapting the standard notion of closedness and consistency [14, 237], we say that an observation table is *closed* if for all $l \in S \cdot I$ there is an $r \in R$ such that r and l are compatible. Closedness ensures that we can create transitions for all inputs in all states.

An observation table is *consistent* if for all pairs of compatible short traces $s, s' \in S$ and all input-output pairs $i \cdot o \in I \cdot O$: either (1) $s \cdot i \cdot o$ and $s' \cdot i \cdot o$ are compatible or (2) $T(s \cdot i)(o) = 0$ or $T(s' \cdot i)(o) = 0$. Consistency requires that the extensions of compatible traces are also compatible. Put differently, if s leads to the same hypothesis state as s', then its extension $s \cdot i \cdot o$ should lead to the same state as $s' \cdot i \cdot o$. This corresponds to the determinism requirement of SMMs.

When an observation table is not closed, there is an $l \in S \cdot I$ such that there is no compatible $r \in R$. We can make an observation table closed by adding such an l violating closedness to the set of short traces S and recalculating the set R. When an observation table is not consistent, there exists a pair of compatible traces s, s', an input-output pair $i \cdot o$, and a column sequence e such that $s \cdot i \cdot o \cdot e \not\approx s' \cdot i \cdot o \cdot e$.

Algorithm 4.2 Ensuring closedness and consistency of an observation table

```
1: function MAKECLOSEDANDCONSISTENT(⟨S, E, T⟩)
2:     if ⟨S, E, T⟩ is not closed then
3:         l ← l' ∈ S · I such that ∀s ∈ S : row(s) ≠ row(l')
4:         S ← S ∪ {l}
5:     else if ⟨S, E, T⟩ is not consistent then
6:         for all s1, s2 ∈ S such that eqRow_E(s1, s2) do
7:             for all i ∈ I, o ∈ O do
8:                 if T(s1 · i)(o) > 0 and ¬eqRow_E(s1 · i · o, s2 · i · o) then
9:                     e ← e' ∈ E such that T(s1 · i · o · e') ≠ T(s2 · i · o · e')
10:                    E ← E ∪ {i · o · e}
11:    return ⟨S, E, T⟩                                  ▷ Closed and consistent obs. table
```

In such a case, we add $i \cdot o \cdot e$ to E. The iterated application of these updates – adding traces to S and sequences to E – eventually establishes closedness and consistency. Algorithm 4.2 depicts the process of making the table closed and consistent. In contrast to deterministic learning, this does not require resampling as compatibility is defined for any amount of samples.

Given a closed and consistent observation table $\langle S, E, T\rangle$ with representatives R, we derive a hypothesis $\mathrm{hyp}(S, E, T) = \langle Q_h, I, O \cup \{\mathit{undef}\}, q_{0h}, \delta_h\rangle$ via:

- $Q_h = R \cup \{q_{\mathit{undef}}\}$ and $q_{0h} = \epsilon$
- For $q \in R$ and $i \in I$, if $\sum_{o\in O} T(q \cdot i)(o) = 0$:
  - $\delta_h(q, i)((q_{\mathit{undef}}, \mathit{undef})) = 1$

  Otherwise:
  - $\delta_h(q, i) = \mu$ where for $o \in O$ if $T(q \cdot i)(o) > 0$:
    $q' = \mathit{rep}(q \cdot i \cdot o)$ and $\mu((q', o)) = \frac{T(q\cdot i)(o)}{\sum_{o'\in O} T(q\cdot i)(o')}$
- If q_{undef} is reachable, then for $i \in I$:
  - $\delta_h(q_{\mathit{undef}}, i)((q_{\mathit{undef}}, \mathit{undef})) = 1$

We create a state for every representative with $\epsilon \in R$ being the initial state. Transitions from $q \in R$ lead to the representatives of the input-output extensions $q \cdot i \cdot o$, where transition probabilities are estimated from T. If we have no observations for an input, we create transitions to a sink state q_{undef}.

Learning Algorithm

Algorithm 4.3 implements the stochastic L^* algorithm for SMMs, adapting L^* for MDPs [237]. In Line 1, we initialize the learning data structures. The main loop starts with a tree query in lines 5 and 6. After updating the learner's data structures, we make the observation table closed and consistent (Line 10) and form a hypothesis $\mathcal{H}$ (Line 11). Given $\mathcal{H}$, we remove table rows and columns that are not needed for hypothesis generation. Line 12 basically removes rows that carry the same information as other rows and cells that do not distinguish rows. For more details, we refer to our previous work [237].

In Line 13, we perform an equivalence query. If it returns a counterexample r, we process it by updating the observation table with information derived from r. L^*-based learning [14, 220] commonly stops once an equivalence query returns *yes*, but we continue learning until the stopping criterion in Line 17 is fulfilled. The reason is that in stochastic learning, we may not be able to find a counterexample given an inaccurate hypothesis that could be improved by additional tree queries. Therefore, we employ a stopping criterion (which we explore in the following section) that takes hypothesis generation into account. Once we stop, we return the final hypothesis.

Algorithm 4.3 The main algorithm implementing L^*_{SMM}

Input: input alphabet I, teacher capable of answering $\mathbf{tq}$ and $\mathbf{eq}$
Output: final learned model $\mathcal{H}$

$S \leftarrow \{\epsilon\}, E \leftarrow I, T \leftarrow \{\}, \mathcal{T} \leftarrow \{\epsilon\}$ ▷ initialize observation table and samples $\mathcal{T}$
$round \leftarrow 0$
repeat
  $round \leftarrow round + 1$
  FTree $\leftarrow$ CREATEIFOPT($\langle S, E, T\rangle$)
  $\mathcal{T} \leftarrow \mathcal{T} \uplus \mathbf{tq}(\mathit{FTree})$
  for all $s \in S \cup S \cdot I, e \in E$ **do**
    $T(s \cdot e) \leftarrow \mathbf{freq}_{\mathcal{T}}(s \cdot e)$ ▷ update observation table
  while $\langle S, E, T\rangle$ not closed or not consistent **do**
    $\langle S, E, T\rangle \leftarrow$ MAKECLOSEDANDCONSISTENT($\langle S, E, T\rangle$)
  $\mathcal{H} \leftarrow \mathrm{hyp}(S, E, T)$ ▷ create hypothesis
  $\langle S, E, T\rangle \leftarrow$ TRIM($\langle S, E, T\rangle, \mathcal{H}$) ▷ remove cells not needed
  $(r, \mathcal{T}_{cex}) \leftarrow \mathbf{eq}(\mathcal{H})$ ▷ Check hypothesis $\mathcal{H}$ against SUL $\mathcal{M}$
  $\mathcal{T} \leftarrow \mathcal{T} \uplus \mathcal{T}_{cex}$
  if $r \neq$ *yes* **then** ▷ we found a counterexample
    $\langle S, E, T\rangle \leftarrow$ PROCESSCEX($r, \langle S, E, T\rangle$)
until STOP($\langle S, E, T\rangle$, $\mathcal{H}$, $round$)
return $\mathcal{H}$ ▷ output final hypothesis

4.3.4 Implementation Details

We implemented Alg. 4.3 in AALPY.[1] AALPY supports learning of both MDPs and SMMs, which we compare empirically in Section 4.4. In the following, we discuss selected aspects of the implementation with a focus on improvements over the original algorithm for MDPs [237], for example, stopping and resampling by the tree query.

SUL Interface. As described in Section 3.4, all active learning algorithms implemented in AALPY rely on an implementation of an `SUL` interface. In the current setting, we consider an SUL implementation that implemented *reset* and *steps* as described in Section 3.4, where *reset* resets the black-box SMM or MDP under learning to an initial state, and the *step* function produces outputs probabilistically based on the current state of the SUL.

Equivalence Queries & Counterexample Processing

The teacher performs two steps in equivalence queries. The first step is checking compatibility between already sampled traces (multiset $\mathcal{T}$ in Algorithm 4.3) and the hypothesis. The second, optional step samples new traces to reveal a counterexample to equivalence between hypothesis and SUL. Sampling happens only when the compatibility check does not reveal a counterexample. This ensures that we use existing samples as efficiently as possible and when there is no counterexample in $\mathcal{T}$ we try to find new counterexamples via sampling. The implementation of these steps follows previous work [237]. We check compatibility between $\mathcal{T}$ and the hypothesis using Eq. 4.1 and we apply random testing for sampling. To ensure that every counterexample can be detected, every input and every trace length has a non-zero probability of being selected during testing.

A counterexample c returned from an equivalence query indicates that the observation table shall be extended in a way to ensure that upcoming hypotheses are correct w.r.t. c. Since hypotheses in active automata learning [14, 237] are generally the smallest models consistent with the queried information, the goal of counterexample processing is to reveal new states.

[1] An interactive example illustrating the learning process is available at `https://github.com/DES-Lab/AALpy/blob/master/notebooks/MDP_and_SMM_Example.ipynb`.

Due to the uncertainties found in the learning process, the counterexample processing might not reveal the additional states in the current learning round. Therefore, before each equivalence query we check whether the last returned counterexample has been successfully processed. If yes, the equivalence query continues, and if not it is reused until it fails to serve as a counterexample.

There are various ways to process counterexamples in L^*-based learning. Our implementation provides three counterexample processing strategies that are commonly applied in deterministic learning.

- *Angluin-style:* Angluin adds all prefixes of a counterexample to S [14].
- *Longest-prefix:* The *longest-prefix* strategy by Shahbaz and Groz [220] splits c into a prefix p and a suffix e, where p is the longest prefix that is already in S. It then adds e to E.
- *Rivest and Schapire (RS)*: The processing strategy proposed by Rivest and Schapire [215] processes a counterexample and finds the shortest suffix e that reveals the difference between the SUL and the current hypothesis. The suffixes of e are added to the E set.

While *RS* and alternative strategies based on the extraction of distinguishing suffixes are very efficient in deterministic learning [8, 109], we found them to be (generally) less efficient in stochastic learning. This is due to the usually low amount of statistical information on counterexample suffixes. Such strategies would require repeated sampling of representative traces from R concatenated with counterexample suffixes until a distinguishing suffix can be found via Eq. 4.1. In contrast, the other two techniques rely solely on sampling performed in subsequent learning rounds.

We want to illustrate the potential sampling cost of a technique, such as *RS* counterexample processing using a hypothetical example. Suppose we have a counterexample of length 16, which we would split into a prefix, a single action a, and a suffix. By replacing the prefix with one of the representative traces and concatenating it again with a and the suffix we might end up with a trace t of length 8. In the case that there are only two outputs with equal probability, it would take on average 256 tries to sample t once. Without going into more details, it should be noted that at the same splitting point, a second trace needs to be sampled and both traces need to be sampled repeatedly for statistical significance. Additionally, a binary search on the counterexample is necessary to determine the correct splitting point. It can be seen that sampling costs would grow fast when individual traces need to be sampled. For the same reason, L^*_{SMM} applies tree queries, where one of the multiple traces gets resampled. If the system under learning is mostly deterministic, sampling a desired trace may be significantly less costly, thus increasing the efficiency of counterexample analysis. In such cases, L^*_{SMM} might further benefit from the technique presented by Howar and Steffen [100], where data structures are extended in a more efficient way in response to counterexamples.

Counterexample length is generally an important factor in the active learning of stochastic systems. Long input-output sequences are hard to sufficiently sample especially when the system under learning is stochastic. This is exacerbated by the fact that counterexample processing techniques from deterministic learning are hardly applicable, as we outlined above. To mitigate this, we sort all the test cases executed during equivalence queries according to their length before counterexample processing. This heuristic helps to identify shorter, easier-to-sample, counterexamples.

Tree Queries

Membership queries in L^* provide information about newly added sequences in the observation table. Tree queries have an analogous purpose. They gather more information on sequences that are in the observation table. While deterministic learning requires a single query (sample) for every sequence, uncertainties affect stochastic learning. We address this issue by sampling traces with the goal of reducing uncertainties.

Algorithm 4.4 *Tree* query

Input: IFOPT *FTree*, SUL with **reset** and **step**
Output: a sampled trace t

1: $node \leftarrow root(\textit{FTree})$, $t \leftarrow \epsilon$ ▷ initialize
2: $\mathbf{reset}()$ ▷ reset SUL
3: **loop**
4: $freqSum \leftarrow \sum_{i \in I} freqLabel(node, i)$ ▷ sum frequencies in IFOPT
5: $inputDist \leftarrow \left\{ i \mapsto p \mid i \in I, p = \frac{freqLabel(node,i)}{freqSum} \right\}$
6: $in \leftarrow \text{CHOOSE}(I, inputDist)$ ▷ choose input
7: $out \leftarrow \mathbf{step}(in)$ ▷ execute SUL and observe output
8: $t \leftarrow t \cdot in \cdot out$ ▷ extend traces
9: **if** $\nexists n \in nodes(\textit{FTree}) : node \xrightarrow{in/out} n$ **then** ▷ did we leave the tree?
10: **return** *trace*
11: $node \leftarrow n$ with $node \xrightarrow{in/out} n$ ▷ walk down one tree level

Uncertainties in stochastic L^* mainly arise from the difference tests and the derived compatibility relation. As discussed in the context of hypothesis generation, this relation is not necessarily an equivalence relation for finite sample sizes. The resulting uncertainties directly affect hypothesis generation, as a trace in $S \cup S \cdot I$ may be compatible to multiple other traces that are not pairwise compatible. Hence, the target state of a transition may be ambiguous. In particular, a trace may be compatible to multiple compatibility class representatives. We devise a sampling strategy with the goal of reducing this form of ambiguity, in order to learn the correct model structure. We start from the viewpoint of the learner and then present the teacher's tree query implementation.

Learner

Given an observation table $\langle S, E, T \rangle$ and $se \in (S \cup S \cdot I) \cdot E$, we assign an uncertainty value *uncert(se)* to se as follows. Let s be the longest prefix trace of se s.t. $s \in S \cup S \cdot I$ and let the number of compatible representatives be

$$cr(s) = |\{r \in R \mid r \approx s\}| \text{ in}$$
$$\textit{uncert(se)} = \max(2 \cdot (cr(s) - 1), 1).$$

The rationale behind *uncert(se)* is that every trace should be compatible with at most one representative. Since we are only interested in the compatibility of rows, we consider the longest trace that labels a row and is a prefix of se. The uncertainty grows with the number of compatible representatives cr, where the multiplication by two puts more weight on this number. We further subtract one, as every trace is compatible with at least one representative in a closed observation table. Finally, we ensure *uncert(se)* is larger than or equal to one for two reasons. We may spuriously conclude that $cr = 1$, i.e., that we are certain that is a unique compatible representative. Due to limited data, this representative may not be the right one corresponding to se. Hence, further sampling should validate if the representative for se is the correct one, which we ensure by having *uncert(se)* ≥ 1. Furthermore, we account for the estimation of transition probabilities as another source of uncertainty affecting hypothesis generation in general. Hence, even if we are certain about the representatives, i.e., the MDP structure, every trace should have a non-zero probability of being sampled. With that, we further improve the accuracy of transition probabilities through extended sampling.

Now, we can define the function `createIFOPT` in Line 5 of Algorithm 4.3.

1. *Trace creation.* Extend the sequences in $(S \cup S \cdot I) \cdot E$ to traces by adding a special output *leaf* $\notin O$ at the end of every sequence, let $Tr = \{s \cdot e \cdot \textit{leaf} \mid s \in S \cup S \cdot I, e \in E\}$.

2. *IOPT creation.* Create an IOPT $Tree$ from the traces in Tr.

3. *IFOPT initialization.* Create an IFOPT *FTree* from $Tree$ by initializing every input frequency with zero.

4. *Adding frequencies.* For each $se \in (S \cup S \cdot I) \cdot E$: add *uncert(se)* to the frequency of every input on the path from the root node to the last edge reached by se.

The frequency label for a given edge ed in *FTree* is the sum of uncertainty values *uncert(se)*, for sequences se traversing ed when starting from the root. Aside from the IFOPT *FTree*, the implementation oof tree queries takes another parameter n_{tree} that defines the number of traces to be sampled by the teacher. We determine n_{tree} proportional to the uncertainty and observation table size via

$$n_{tree} = \left\lfloor \frac{\sum_{se \in (S \cup S \cdot I) \cdot E} \textit{uncert(se)}}{2} \right\rfloor . \tag{4.4}$$

Roughly speaking, we take one sample for every unambiguous cell in the table and additional samples for ambiguity.

Teacher

The teacher performs tree queries by sampling n_{tree} traces corresponding to directed random walks on the IFOPT *FTree* created by the learner. Algorithm 4.4 implements this form of sampling. It starts with an initialization in lines 1 and 2. The sample loop starts with the selection of an input in in lines 4 to 6, where the selection probability of in is proportional to the frequency assigned to in. Here, we use $\text{CHOOSE}(I, d)$ to sample an input in I according to a probability distribution d. Next, we execute in on the SUL and extend the sampled trace t with in and the SUL output. When there is no path in the IFOPT corresponding to t, we return t. This is guaranteed to happen when reaching a leaf, as leaves are labeled with a symbol not in the output alphabet.

Stopping

Similarly to tree queries, stopping takes ambiguity into account. We stop, when ambiguity decreases. For stopping, we quantify ambiguity or rather the absence thereof as the number of row traces that have a single compatible representative. This number $unambiguity$ is given by:

$$unambiguity = \frac{|\{s \in S \cup S \cdot I \mid cr(s) = 1\}|}{|S \cup S \cdot I|}$$

We also used this value to decide when to stop learning in our previous work [237]. Previously, we stopped learning once $unambiguity$ was greater than a fixed threshold. However, we concluded from experiments that a fixed threshold is not an ideal choice for SMM learning concerning efficiency. In these experiments, we measured hypothesis accuracy in relation to the value of $unambiguity$ during the course of learning. For this purpose, we quantified accuracy as the average error made in model checking computations, as compared to model checking performed on the true model of the SUL. We observed a general trend that the hypothesis accuracy converges at the same time when $unambiguity$ converges to a plateau.

Figure 4.2 shows plots of the $unambiguity$ value, the maximum error in various model checking computations, and the corresponding average error on two examples that we discuss in more detail in Section 4.4. The x-axis displays the number of learning rounds. We can see that upon reaching an $unambiguity$ value of approximately 0.8, the maximum and average errors are close to zero and stay close to zero. At this point, further learning and sampling costs resources, but does not contribute to the accuracy of the model. For this reason, we stop learning when we detect that $unambiguity$ reaches a plateau, i.e., the difference between several consecutive $unambiguity$ values is below a small positive ϵ.

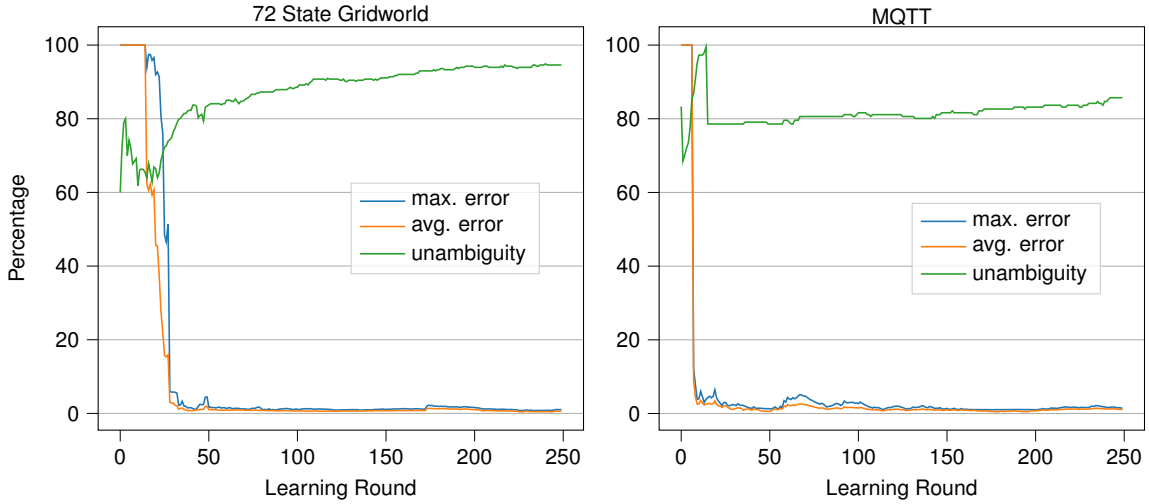


Figure 4.2: Relation between number of unambiguous rows and hypothesis accuracy

In common with previous work [237], we do not stop when q_{undef} is reachable in the hypothesis. This state is reachable, when there is a state-action pair for which there is no information at all. Additionally, it is possible to specify a minimum number of rounds and a maximum of rounds. The early stopping criterion can also be disabled in favor of a fixed threshold.

4.3.5 Analysis of L^*_{SMM}

Complexity

In the following, we will analyze the complexity of individual operations performed by L^*_{SMM}, such as queries. For this purpose, let m be the length of the longest sampled trace, let n be the number of sampled steps, and let k be the number of different sampled traces. In the worst case, k grows linearly in n. For simplicity, we consider the set of sampled traces to be prefix-closed, because whenever we observe a trace, we observe all its prefixes as well.

Since we add only traces to S that have been sampled at least once, we can bound the length of sequences in S by m and the cardinality of S by k. We can analogously bound the length of sequences in E and the cardinality of E, as E contains suffixes of sampled traces. Making an observation table closed and consistent requires time in $O(k^3 \cdot |O|)$. Checking consistency requires iterating over all pairs of rows and checking compatibility for each cell. There are at most k^2 row pairs and at most k columns (cells in each rows) and each compatibility check requires $|O|$ computations (see Eq. 4.1). Additionally checking the extensions of compatible rows only adds a constant factor. Hence, the runtime is in $O(k^3 \cdot |O|)$. Fixing a consistency violation simply amounts to adding a new element to the E set. Checking closedness requires compatibility checks between every pair consisting of a long and a short row. Thus, closedness checks are in $O(k^3 \cdot |O|)$ as well. Creating a hypothesis from an observation table is in $O(k \cdot |I| \cdot |O|)$. We need to potentially create a transition for every input-output pair from every state. There are $|I| \cdot |O|$ such pairs and there are at most k states.

The IFOPT creation for tree queries takes time in $O(k \cdot m)$, as every unique trace of length at most m is added at most once. The value n_{tree} of traces sampled during a tree query (see Eq. 4.4) is at most k^2, but generally much lower. Hence, a tree query performs at most $k^2 \cdot m$ sampling steps.

Equivalence queries consist of two steps, checking compatibility and sampling traces to actively check for equivalence between hypothesis and SUL. The amount of sampling can be adjusted freely according to the sampling budget and accuracy requirement. The compatibility check between sampled information and hypothesis requires runtime in $O(k \cdot m \cdot |O|)$. For every sampled trace t we determine a hypothesis state and its representative trace r, which has a length of at most m. For t and r, we

perform a compatibility check, which takes time linear in the number of outputs. This analysis matches our experience in that making observation tables closed and consistent takes the most time. While the analysis of sampling steps performed by tree queries provides a conservative upper bound, we observe that tree queries perform thorough sampling leading to accurate models.

Finally, since we use a heuristic to decide stopping, we cannot state a complexity bound on entire runs of L^*_{SMM}.

Convergence

Under mild, common assumptions on equivalence queries (every trace must have a non-zero probability to be sampled), the learned model converges in the limit, i.e., with stopping disabled, to a minimal SMM equivalent to the SUL. A proof of convergence can be adapted from [237]. In the next section, we empirically analyze the accuracy of models learned from finitely many traces.

4.4 Empirical Evaluation

In order to evaluate our method, we conducted a study considering six different stochastic systems that have been used also in previous work [4, 237]. More concretely, we simulated known models of stochastic reactive systems through a *reset* and a *step* operation, thus treating them as black boxes. After learning models from the simulations, we measured the accuracy of the learned models. As a measure of accuracy, we used the absolute difference between the probabilities computed by probabilistic model checking properties on a learned model and the corresponding known true model. In the following, we report this form of error averaged over several properties for each system.

Systems used as a basis for learning are: a 35-state and a 72-state *Gridworld* [237], the *slot-machine* previously used in [143, 144], parts of the *MQTT* protocol encoded as a 61-state MDP, parts of the *TCP* protocol encoded as a 151-state MDP, and a *Bluetooth Low-Energy* device encoded as a 156-state MDP. For the first three systems, we used model checking properties also used in previous work [237] to enable a direct comparison. We derived the *MQTT*, *TCP*, and *Bluetooth Low-Energy* models from learned deterministic models [74, 196, 234] by injecting stochastic faults into the model, denoted as a "crash" state (see also [4]). More concretely, we added probabilistic behavior at selected places in the deterministic models, such that previously deterministic transitions now have a chance of leading to a "crash" state. For this purpose, we randomly selected a subset of the transitions in the deterministic models and split each of them into two transitions. One transition of each such pair produces the original, correct output and occurs with a probability $1 - p_c$, whereas the other one produces a newly introduced $crash$ output and occurs with probability p_c, leading to a "crash" state. This allows us to efficiently simulate network protocols and allow us to reason about the applicability of L^*_{SMM} in real-world scenarios. Throughout the remainder of this section, we will reason about the benchmark model's underlying structure and how it affects learning performance.

To verify the accuracy of the learned models, we derived a set of reachability properties for each model. More precisely, those properties check for the bounded reachability of reaching a fault or a goal state in the models with varying bounds. Correct property values were computed on the original model and on the learned models, with the error being the absolute difference between the results. To compute the values, we used probabilistic model checking with PRISM and statistical model checking [131]. The former computes the maximum probability of satisfying the property, quantified over all policies for choosing actions. For this reason, we denote the results with $\mathbb{P}_{MAX}$. For statistical model checking, we sample actions according to the uniform distribution and estimate the probability of satisfying properties under such a random policy. Therefore, we denote the results with $\mathbb{P}_{RAND}$. Concretely, we apply a Monte Carlo simulation for the estimation with a Chernoff bound [183] with an error bound of 0.01 and a confidence of 0.995. We use statistical model checking as a means of getting an insight into

Table 4.2: Comparison of MDP and SMM learning

	Learning Time (s)			# Traces		
	MDP	SMM	Improvement	MDP	SMM	Improvement
35-State Grid	8.61	5.67	51.85%	68736	44954	52.9%
72-State Grid	40.92	29.85	37.09%	199789	113690	75.73%
MQTT	8.59	6.97	23.24%	68964	39904	72.82%
TCP	21.94	13.4	63.73%	98333	54514	80.38%
Bluetooth	17.26	6.98	60.91%	94525	35605	165.48%
Slot Machine	60.65	306.4	-80.21%	456243	622285	-26.86%

Table 4.3: Results for learning models of the 72-state Gridworld

	true	MDP	SMM	Error MDP	Error SMM	Improvement
# Traces	-	199789	113690	-	-	75.73%
# Steps	-	1363233	762338	-	-	78.82%
$\mathbb{P}_{max}(F^{\leq 14}(goal))$	0.9348	0.931	0.94	0.0038	0.0052	-26.92%
$\mathbb{P}_{max}(F^{\leq 12}(goal))$	0.6712	0.6943	0.681	0.0231	0.0098	135.71%
$\mathbb{P}_{max}(\neg M\ U^{\leq 18}(goal))$	0.9743	0.9721	0.973	0.0022	0.0013	69.23%
$\mathbb{P}_{max}(\neg S\ U^{\leq 20}(goal))$	0.1424	0.2138	0.1482	0.0714	0.0058	1131.03%

the accuracy of learned models for longer traces, as maximum probabilities computed via probabilistic model checking often equal or are close to 1 for long traces.

All experiments have been performed with our implementation in the open-source library AALpy v1.2.7 running on a Dell Latitude 5410 with an Intel Core i7-10610U processor, 8 GB of RAM running Windows 10 and using Python 3.9. We configure the Hoeffding-bound-based difference check with a constant $\alpha = 0.05$ and we limit the number of traces sampled during equivalence queries by $n_{cextraces} = 150$. In comparison to previous work [237], we reduced the number of configurable parameters, while achieving robust learning performance. In addition, all benchmark models can be found on AALpy's official repository[2]. Due to the stochastic nature of the experiments, we repeated every experiment, consisting of learning and model checking, 20 times and report average results.

4.4.1 Comparing MDP and SMM Learning

The contributions made in this work are twofold. Firstly, a formalization of SMMs and adaptation of L^*_{MDP} that is able to learn them, and an overall algorithmic improvement of the original algorithm. In Section 4.4.1 to Section 4.4.5 we focus on the comparison of MDP and SMM learning with our new and improved algorithm, while in the Section 4.4.6 we compare L^*_{SMM} with its predecessors.

To compare the performance of MDP and SMM learning, we measured the learning time and the number of traces sampled by the teacher. While total learning time can be used to evaluate the feasibility of our approach in simulated environments, it is important to note that in real-world non-simulated environments, sampling of traces, i.e., the interaction with the SUL, is the most time-consuming aspect of active automata learning. Table 4.2 summarizes the results of our benchmarking study. We see that for all examples but *slot machine* SMMs are the preferred formalism.

Table 4.3 shows detailed results of learning MDPs and SMMs of the 72-state *Gridworld*. In this example, learning SMMs reduces the state space by about 75%, as the SUL can be modeled by a 42-state SMM. This size difference accounts for the better and faster learning of SMMs compared to MDPs. Note

[2] https://github.com/DES-Lab/AALpy

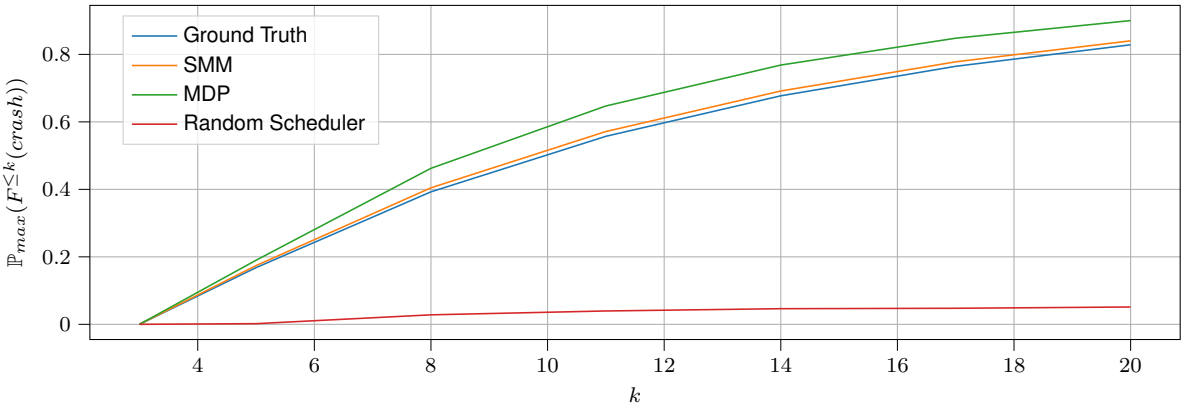


Figure 4.3: Maximum probabilities of reaching a "crash" state of the Bluetooth Low-Energy model on ground-truth model and learned models within a bounded number of steps

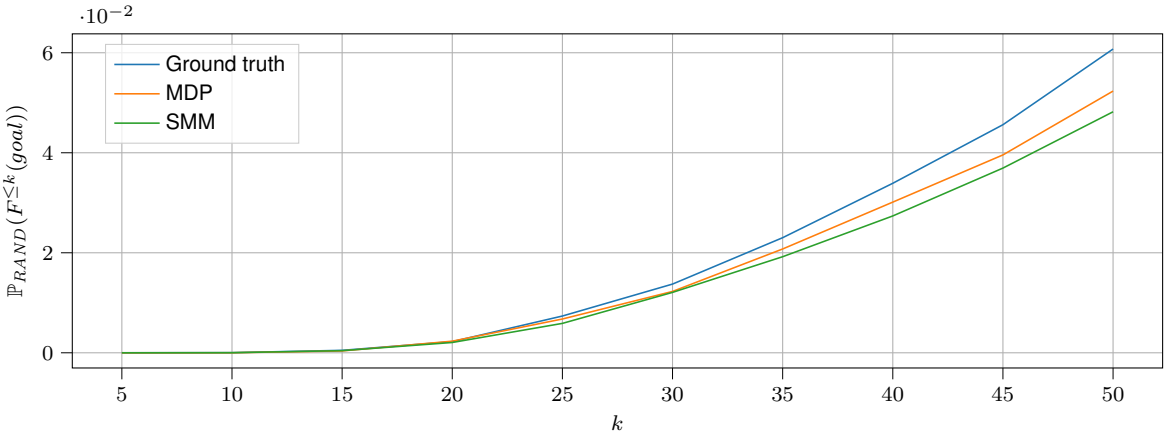


Figure 4.4: Probability of reaching a "goal" state in the 72 state Gridworld within k steps. Probabilities are computed using Monte Carlo simulation with a random scheduler

that on average both learned the underlying model accurately, with the maximum average error for MDP learning being 7.1% and 0.9% for SMM learning.

Figures 4.3 and 4.4 depict the effect of the property bound in the reachability analysis, both with a property-based model checking (Figure 4.3) and with statistical model checking [131] (Figure 4.4). These experiments show that learned models remain accurate with the increasing property bound, meaning that the learned model remains true to the system-under-learning even for longer, harder-to-sample, test sequences.

More concretely, Figure 4.3 depicts the value of reaching the "crash" state in the *Bluetooth Low-Energy* model. We observe that the accuracy of learned models remains consistent with the increasing bound, with learned SMM being slightly more precise (interestingly, differences in absolute errors between learned MDP and SMM models are, while small, statistically significant, according to the p-value of Student's t-test [229]) than the learned MDP while requiring significantly fewer learning queries (as shown in Table 4.2). Figure 4.4 depicts the probability of reaching a "goal" state on the 72-state Gridworld with an increasing step bound. These results are consistent with those shown in Figure 4.3, in that the accuracy of learned models remains high with the increasing property bound. However, in this case learned the MDP is statistically significantly more accurate (with respect to the absolute error) than learned SMM, albeit with higher learning costs.

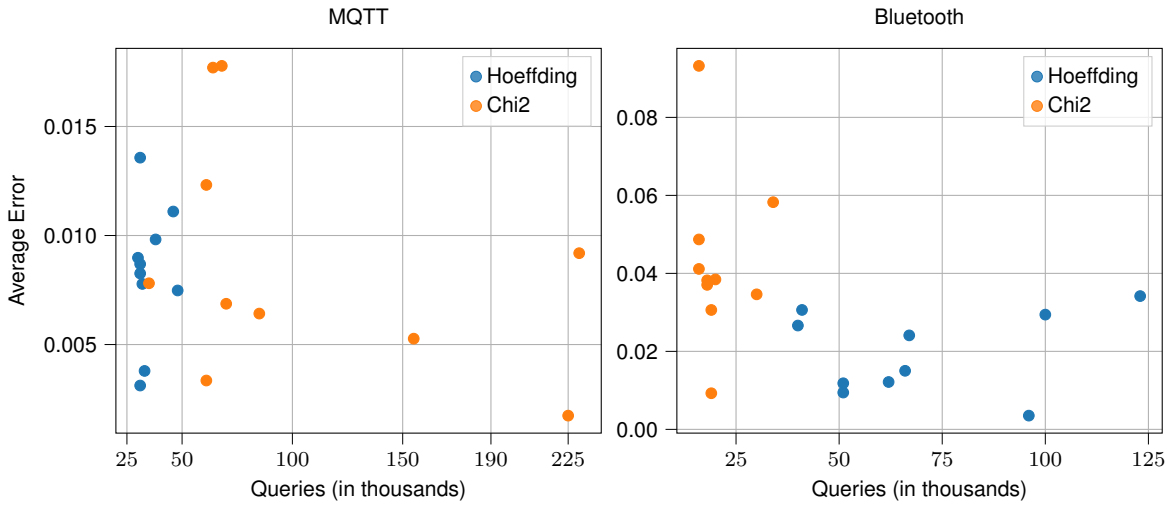


Figure 4.5: Accuracy and learning costs comparison between Hoeffding and χ^2 statistical compatibility checks

Table 4.4: Comparison of the impact of different counterexample processing strategies on the total number of sampled traces during learning

Counterexample Processing	35-State Gridworld	72-State Gridworld	MQTT	TCP	Bluetooth	Slot Machine
Angluin-style	39718	139358	30761	54514	41237	564368
Longest Prefix	77974	254087	26493	33436	24499	629069
RS	51802	147297	45091	47519	24188	1083828

4.4.2 Comparison of Statistical Compatibility Checks

In Section 4.3.1 we outlined two statistical compatibility tests to determine whether the two traces are equivalent. We compare the effect of the different statistical compatibility checks on the overall L^*_{SMM} learning process. In both compatibility checks, α was set to 0.001, increasing the confidence in the verdicts of the compatibility checks. With all other settings being constant, we repeated the learning process multiple times for each statistical compatibility test and compared the number of traces executed during learning and the average error over all properties.

Figure 4.5 depicts the results of this comparison. Experiments were repeated for MQTT and Bluetooth Low-Energy models. Firstly, we observed that both statistical compatibility checks lead to accurate models, with a maximum average error being 1.5% and 9%. From the results, we make the following observations: (1) more queries do not necessarily lead to more accurate models and (2) there is more variability in the learning results when using χ^2 compatibility checks. Hoeffding compatibility checks appear to be slightly more preferable, given that it tends to lead to a lower learning error while keeping the learning costs low.

In addition, these findings show that the learning algorithm is not dependent on the Hoeffding compatibility check, but can be used with other statistical compatibility checks as well. In future work, it would be interesting to implement and further study the effects of χ^2 and other statistical compatibility tests on other stochastic automata learning algorithms, such as IOALERGIA.

4.4.3 Comparison of Counterexample Processing Strategy

In Section 4.3.4 we outlined three counterexample processing strategies. To evaluate their effect on the learning process, we compared them on the previously described benchmark models. As stated in

Table 4.5: Comparison between SMM and MDP learning costs on randomly generated MDPs and SMMs. Cell values show the difference between total learning costs, expressed as a difference between the number of queries posed by SMM and MDP learning. Positive cell values indicate that SMM learning was more cost-efficient than MDP learning for a specific experiment, and vice versa.

Random Stochastic Mealy Machine

		Number of States					
Input Alphabet Size	Output Alphabet Size	5	10	15	20	30	50
3	7	69.51%	83.28%	52.91	58.12%	79.63%	70.97%
4	10	86.97%	83.93%	86.78%	78.31%	76.34%	82.55%
7	15	93.38%	95.18%	94.35%	94.85%	96.18%	87.32%

Random Markov Decision Process

		Number of States					
Input Alphabet Size	Output Alphabet Size	5	10	15	20	30	50
3	7	-11.76%	-19.58%	-22.6%	-59.59%	–43.86%	-73.25%
4	10	18.06%	19.78%	37.88%	-29.27%	-17.11%	-20.54%
7	15	-37.27%	-44.41%	-65.86%	-45.11%	-19.78%	-0.09%

Section 4.3.4, we sort test cases executed by an equivalence query by length, so that we increase the probability of finding short counterexamples.

Results from the counterexample processing evaluation show that the selection of a counterexample processing strategy in a stochastic setting influences the sampling efficiency of the learning process. However, the selection of an appropriate counterexample processing strategy is influenced by the topology and transition probabilities of the SUL, which are unknown in a black-box setting. Therefore, we propose the usage of the *Angluin-style* counterexample processing, as it consistently performed well without major performance outliers. The other two methods, especially *Rivest and Schapire* could be more appropriate in settings where we know apriori that the majority of SUL transitions are deterministic. This is an important consideration as *Longest-prefix* and *Rivest and Schapire* add suffixes to the E set of the observation table, therefore requiring fewer, but significantly longer queries compared to *Angluin-style*, which adds prefixes of the counterexamples to the S set. Such long queries are in turn hard to sample, which usually prolongs the learning process.

More concretely, Table 4.4 shows the influence of the counterexample processing on the total number of traces sampled during the learning and equivalence-checking process. From the experimental results, which are averaged over multiple execution runs, we observe that the *Angluin-style* counterexample processing is consistently relatively sample efficient without major outliers, while the *Longest-prefix* counterexample processing performed better on network protocol benchmarks (MQTT, TCP, Bluetooth). Although the performance of *Rivest and Schapire* is acceptable for network protocols, we observed that it might lead to an increase in total number of traces sampled for other benchmarks, mostly due to additional sampling required by *Rivest and Schapire*. Note that the accuracy of learned models was high, regardless of the counterexample processing method, implying that the selection of the counterexample processing method does not impede convergence to the true model.

4.4.4 Comparison on Randomly Generated Automata

To further examine the difference between the learning processes of MDP and SMM learning, we performed a set of experiments on randomly generated automata. Namely, we randomly generated both MDPs and SMMs and evaluated whether the type of the system under learning affects the learning process. Randomly generated automata were scaled along three dimensions: the number of states of the

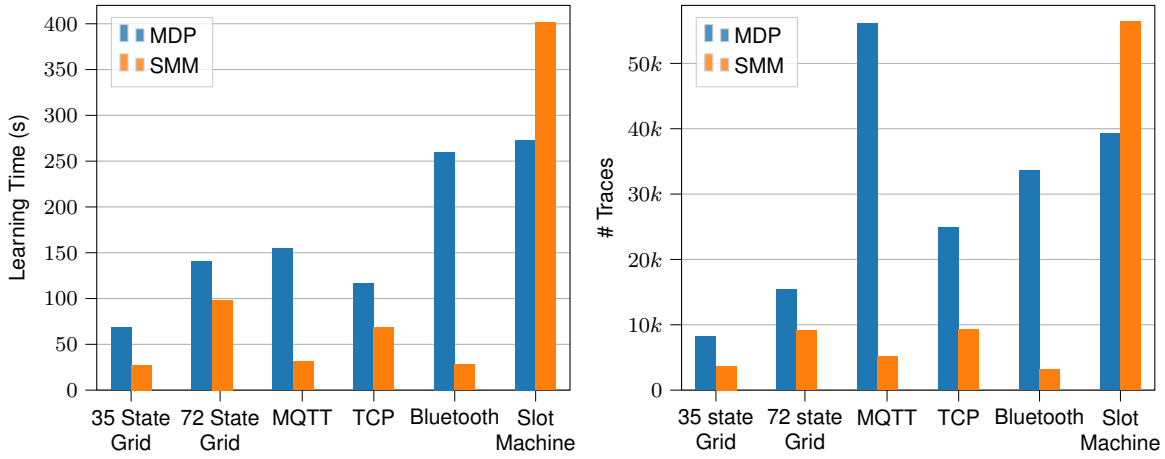


Figure 4.6: Learning time in seconds (left) and number of traces (right) needed to reach at most 2% error for all properties

automata, input alphabet size, and output alphabet size. For each parameter configuration, we generated a random MDP and a random SMM.

Table 4.5 summarizes the comparison of MDP and SMM learning on randomly generated automata. The first result block compares the SMM and MDP learning of randomly generated SMMs. We observe that SMM learning consistently outperforms MDP learning if the system-under-learning has an SMM-like structure. For example, a cell value of 95.18% (for randomly generated 10-state SMM with the input alphabet size of 7 and output alphabet size of 15) indicates that the SMM learning required 95.18% fewer queries than the MDP learning of the same random SMM. More concretely, SMM learning required only 19k input-output traces, while MDP learning required 398k input-output traces, making SMM learning 20 times more efficient for this particular example. On the other hand, as seen in the lower half of Table 4.5, if the randomly generated automaton is an MDP, SMM learning required up to 73% more traces.

These findings indicate that the structure of the system under learning can significantly affect the learning process and required costs. Given that the L^*_{SMM} has no insight into the system-under-learning structure, a designer has to make a qualitative decision about whether to choose MDP or SMM learning. Based on our experience, we postulate that SMM learning might be preferable for most real-world systems due to SMM's more compact formalism.

4.4.5 Convergence

To evaluate learning speed, we disabled stopping and checked how long it takes for the learned hypothesis to reach a certain accuracy. For this purpose, we learned until creating a hypothesis with a probabilistic model checking error of at most 2% for all properties defined for the respective example. Figure 4.6 summarizes the results of these experiments. Notice in Figure 4.2 that the error stays close to zero after a certain number of rounds. Hence, these experiments serve to estimate how long it takes to converge to an accurate model.

All examples but *slot-machine* show noticeable improvements in both learning time and more importantly required traces when comparing MDP and SMM learning. The largest improvements between MDP and SMM learning are noticeable for *MQTT*, *TCP* and *Bluetooth* models. In those examples, models can be more compactly encoded as SMMs than as MDPs, which leads to a significant decrease of learning costs when compared to MDP learning. On the other hand, the structure of the *slot-machine* favors MDP learning. Learning a *slot-machine* MDP, required 25% fewer traces than learning an SMM. We argue that the good performance of MDP learning results from the fact that the minimal slot-machine MDP is only slightly larger than the minimal SMM and that many MDP states can be distinguished solely

Table 4.6: Comparison of L^*_{SMM} with original L^*_{MDP} and IOALERGIA

Algorithm	L^*_{SMM}		Classic L^*_{MDP}		IOALERGIA	
	# Traces	Mean Error	# Traces	Mean Error	# Traces	Mean Error
35-State Grid	44954	0.017	391, 530	0.0113	391, 530	0.667
72-State Grid	113690	0.04	515,950	0.069	515,950	0.163
MQTT	39904	0.015	300500	0.015	300500	0.018
TCP	54514	0.005	279423	0.004	279423	0.19
Bluetooth	35605	0.011	317446	0.016	317446	0.10
Slot Machine	622285	0.011	1,567,487	0.015	1,567,487	0.35

based on their output label without statistical tests. In general, when the output alphabet size is large, then MDP learning may be more efficient. In the extreme case, where every state is labeled with a unique output, the model structure is already given and learning amounts to estimating transition probabilities.

4.4.6 Comparison with Related Work

To put the presented improvements in context, we will compare the results found in this chapter with two stochastic automata learning algorithms: L^*_{MDP} [237] and IOALERGIA [143, 144]. The algorithm presented in [237] serves as a basis for L^*_{SMM}, and we can directly compare results from overlapping benchmark models (First Grid, Second Grid, Slot Machine). For other benchmark models, we have obtained new results with AALPY, which implements both the original version of the L^*_{MDP} and IOALERGIA. It is important to note that by "original" L^*_{MDP} we consider the version of the algorithm presented in [237], without any of the optimizations presented in this chapter, and when we were comparing SMM and MDP learning in the previous sections, we used the version of the algorithm presented in this chapter.

Table 4.6 shows the results for the three learning algorithms on all benchmark models. Following the comparison method performed in [237], IOALERGIA was given the same number of random traces as L^*_{MDP} required throughout the learning. From the results shown in Table 4.6, we conclude that L^*_{SMM} requires significantly fewer traces than the original L^*_{MDP} , while maintaining high accuracy. Interestingly, in a few examples, it even has marginally better accuracy than L^*_{MDP} , but with almost an order of magnitude fewer traces. However, it is important to note that the accuracy differences between both algorithms are quite small, in some cases negligible. Compared to both algorithms, IOALERGIA has worse accuracy, even with a high number of provided traces.

The difference in the number of traces required by L^*_{SMM} and the original L^*_{MDP} to learn accurate models can be attributed to several optimizations presented in this chapter. Firstly, we introduced an additional stopping criterion in Section 4.3.4, which stops learning when we observe that additional sampling does not improve the ambiguity of the observation table. Secondly, we have replaced the completeness query and the frequency query with the tree query (Section 4.3.4), which makes the algorithm automatically select the number of queries it asks during each learning round. Finally, counterexample processing techniques and our strategy to invest testing resources in finding the shortest counterexamples makes the algorithm more efficient, as outlined in Section 4.3.4.

4.5 Passive Learning of Stochastic Mealy Machines

Finally, we briefly reason how the standard MDP variant of IOALERGIA can be adapted to learn SMMs. When constructing an IOFPT for passive SMM learning we start by assigning the input-output pairs to transitions of the IOFPT, in contrast to the MDP variant which assigns inputs to the transitions and outputs to the IOFPT nodes. As explained in Section 2.3.2, IOALERGIA merges two nodes of an IOFPT if they share the same output and their futures pass the statistical compatibility test. To adapt IOALERGIA to SMMs, we simply ignore the first requirement in the IOFPT node compatibility, that is the two nodes

Table 4.7: Comparison of MDP and SMM variants of IOALERGIA

	35-State Gridworld	72-State Gridworld	MQTT	TCP	Bluetooth	Slot Machine
MDP Mean Error (%)	0.529	0.6573	0.014	0.2148	0.1246	0.0395
SMM Mean Error (%)	0.644	0.6529	0.008	0.0948	0.2606	0.0632
SMM Improvement (%)	-0.1145	0.004	0.006	0.12	-0.1360	-0.0237

share the same output. This is a consequence of different output functions, as in MDPs outputs are tied to the states, and in SMMs to state-input pairs. The remainder of the algorithm remains unchanged, and at the end of the execution, the resulting IOFPT is encoded as SMM.

We evaluate IOALERGIA for MDPs and SMMs on the previously used benchmark models. For all models, we randomly generated $250k$ sequences of length [10-30]. We used the same input-output traces for IOALERGIA to learn MDPs and SMMs from the provided data. Table 4.7 shows the result of this comparison. We observe that the MDP variant of IOALERGIA has yielded more accurate models for two out of six benchmarks, while the SMM variant was more accurate on one benchmark model. On the remaining three benchmarks both variants performed similarly.

The initial evaluation of the SMM variant of IOALERGIA suggests that the passive learning of SMMs does not yield as many practical benefits as the active learning of SMMs. While the smaller model size greatly improves the active automata learning process, in the passive learning context the switch from MDP to SMM does not seem to provide as many practical benefits. We speculate that one of the reasons why the SMM variant of IOALERGIA is sometimes less accurate than the MDP variant is due to the relaxed IOFPT tree node merging procedure. That is, a requirement that two nodes in IOFPT share the same output helps the MDP variant of IOALERGIA to further better differentiate states, while the SMM variant might be more eager to deem two nodes compatible solely based on the statistical compatibilities of their continuations.

4.6 Concluding remarks

We presented L^*_{SMM}, an L^*-based algorithm for active learning of models of stochastic reactive systems. By improving previous work [237] and adapting it from learning MDPs to SMMs we can learn stochastic models more efficiently while achieving accurate results. The experimental evaluation of our implementation shows a significant reduction in the number of required system interactions. Since interactions with the system are typically the time-consuming aspect of applications of automata learning, this number is the most important efficiency metric. In particular, we reduced the required number of system traces, i.e., sequences of interactions, by up to 8.7 times, as compared to MDP learning.

RQ 1.1 How can we improve upon the state of the art in modeling of stochastic systems?

L^*_{SMM} presents several improvements over the existing state-of-the-art. The most notable difference that improves the efficiency of the automata learning process is the choice to switch the learning formalism from MDPs to SMMs. Other algorithmic improvements such as early stopping, counterexample processing, and the implementation of tree queries further improve upon the state of the art, while simplifying the algorithm. Furthermore, since L^*_{SMM} uses a variety of heuristics to determine the sampling strategy, it is less dependent on user parameterization compared to the original L^*_{MDP}. Finally, from our empirical evaluation, we conclude that L^*_{SMM} yields models of equal or even improved accuracy compared to L^*_{MDP}, while requiring substantially fewer interactions with the system.

5

BLACK-BOX EXTRACTION OF FINITE STATE MODELS FROM RNNS

Declaration of Sources

This chapter is based on our work titled "Learning Finite State Models from Recurrent Neural Networks"[167] and presented at iFM2021. This work was slightly extended in "Testing-based Black-box Extraction of Simple Models from RNNs and Transformers" [168] and presented at ICGI2023. The proposed method was used to participate in the TAYSIR competition [69] on the extraction of simple and interpretable models from RNNs and transformers, in which we won the first place.

5.1 Motivation

Recurrent neural networks (RNNs), introduced in Section 2.5, are a type of neural network used to model sequential data. Due to their stateful nature, RNNs are ideal for challenging tasks on high-dimensional sequential data, such as natural language processing [232, 268], time series forecasting [67], and reinforcement learning in partially observable environments [92], where agents take actions based on a history of events. However, their internal decision-making process remains ambiguous. What is more, reasoning about the RNN internal decision-making process is especially challenging since we can hardly reason about the training data itself. In other words, we lack an interpretable ground-truth model that captures the complexities of the problem that RNN aims to solve.

To that end, many researchers have turned to training of RNNs on formal languages to better understand the internal processes of RNNs. The usage of formal languages as a ground truth enables researchers to draw conclusions about an RNN's behavior with respect to the formal structure of the model used for data generation. In particular, the ability of RNNs to learn regular languages prompted research on extracting formal models that explain the internal mechanisms underlying trained RNNs [186].

More recently, we have seen renewed interest in extracting finite-state models from RNNs [69, 145, 185, 271]. Such models enable a manual inspection of RNN behavior on one hand, and the application of automated verification techniques like model checking on the other hand — both would be practically impossible otherwise. Automata learning-based approaches are a natural fit in this context since they concern themself with the extraction of finite-state models from black-box systems. In the context of

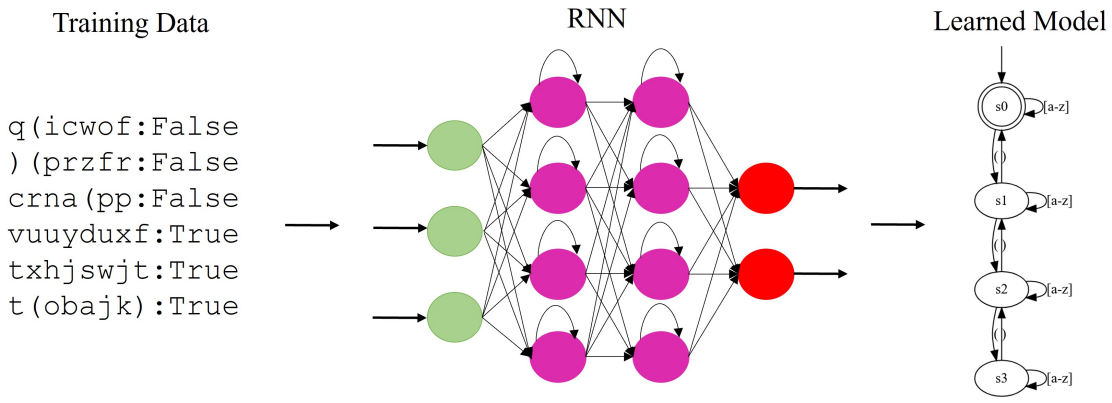


Figure 5.1: Abstract visualization of the RNN training and model extraction pipeline

this thesis, automata learning allows us to extract models from RNNs while treating them as a black-box, therefore ignoring their inherent complexities while only focusing on their input-output behavior.

In the remainder of this chapter, we will examine such automata-learning-based extraction approaches. Firstly, we discuss how to learn finite-state models from RNNs trained on regular languages by applying active automata learning to analyze their input-output behavior. In this context, we propose to apply model-guided conformance testing in combination with active automata learning, which is applicable in a black-box setting. We compare this approach with two other approaches based on active automata learning, which have been applied to extract FSMs from RNNs; one white-box [271] and one black-box approach [145]. Since models should faithfully model the system that they represent, we examine the accuracy of the learned models. We compare the three different approaches based on the properties of the learned models and of the learning process.

In line with research in this area [145, 185, 271], we train RNNs on words sampled from regular languages that can be represented by small finite automata. Finite automata as ground truth enable a manual analysis while following a black-box approach enables scaling to RNNs of different sizes and architectures. Figure 5.1 abstractly illustrates the learning pipeline that we consider in this chapter: an RNN is trained from samples generated by some regular language, and automata learning is used to extract a finite-state model from the trained RNNs. This pipeline allows us to reason about the RNN's accuracy since we have knowledge about the structure of the ground-truth model and the structure of generated RNN training data.

5.1.1 Guiding Idea

In Section 2.2 we outlined the principles and working method behind active automata learning. While active automata learning algorithms might vary in the efficiency of the model-learning process itself, they will all return the same model if the equivalence oracle is able to find all counterexamples between the inferred hypothesis and the SUL. In other words, the outcome of automata learning algorithms depends on the "strength" of the used equivalence oracle, that is, its ability to correctly judge whether the current hypothesis conforms to the SUL. Upon closer examination of the current state of the art in the model extraction from RNNs [145, 271], we identified a need for an automata-learning-based extraction approach that puts special emphasis on a strong equivalence query from a testing perspective.

Structure. This chapter is structured as follows: in Section 5.2 we propose our method and elaborate on how it differs from other model-extraction approaches [145, 271]. We then perform an experimental evaluation against competing approaches in Section 5.3. In addition, we also show how our method could be extended to model RNNs trained on regression tasks and present our results on the benchmarks

used in the TAYSIR competition. We present concluding remarks and position this work in the scope of the thesis in Section 5.5.

5.2 Method

In this section, we outline how automata learning could be used to mine models that capture the input-output behavior of RNNs trained to recognize regular languages.

As RNNs have an unbounded input space of sequences, active automata learning is well-suited to handle such input data and thus to extract models from the input-output behavior of RNNs. What is more, it provides guidance for exploring this input space through input queries and conformance testing provides guidance for exploration during equivalence queries. To avoid confusion between automata learning and training of the RNNs, we will exclusively use the word learning in the context of automata learning and training in the context of RNNs.

We start by outlining some of the practical aspects of model mining from RNNs trained on regular languages. We then cover the role of the equivalence query in this process. Since we postulate that the equivalence query is the deciding factor in the model mining process, we formulate two research questions that will help us perform a comparison of the proposed approach with state-of-the-art.

5.2.1 SUL Interface

As outlined in Sect. 3.4, the active automata learning setup could be divided into three phases: implementation of the SUL interface, selection and parameterization of an equivalence oracle, and setup of the learning algorithm.

SUL Interface. We observe that these operations correspond to the operations that are required to execute an input query on the SUL: *step* and *reset*. To execute an input query, we first reset the RNN's internal state to a fixed value and then perform a sequence of steps. Each step on the RNN is executed with an element of the input alphabet and the current hidden state, and an RNN output and the updated hidden state are returned. If the RNN framework does not support the step-by-step creation of variable-length sequences, then the whole input sequence can be passed to the RNN, and the output associated with the whole input sequence can be obtained.

With AALPY we can easily use the majority of frameworks that provide standardized RNN implementations as an SUL. Listing 5.1 provides a conceptual representation of how the SUL interface is used to learn RNNs with AALPY. Concrete implementations of SUL interfaces that follow the same basic method but consider framework-specific constraints can be seen in our publicly available repositories [1] [2].

Input and Output Abstraction. There is no need for additional input and output abstraction for RNNs trained on regular languages, as the networks themself attempt to mimic the behavior of the ground truth regular language. Framework-specific transformations of the elements of the input alphabet to the format used by an RNN are required, but we do not consider those as an abstraction. More concretely, we used one-hot encoding. One-hot-encoding is used to represent discrete elements of an input alphabet as vectors where all values except one element are set to zero except for one value which is set to one. Since RNNs were trained on regular languages, outputs are in the discrete domain and no additional output abstraction is necessary. However, in the scope of the TAYSIR competition, we also considered RNNs trained on language modeling tasks, and since the output of these RNNs is not discrete we introduce an appropriate abstraction method in Sect. 5.4.2.

[1] https://github.com/DES-Lab/Extracting-FSM-From-RNNs/blob/master/RNN_SULs.py

[2] https://github.com/emuskardin/taysir_competition_mbt/blob/master/SULs.py

Listing 5.1: Abstract representation of RNN `SUL` implementation in AALPY

```
from aalpy.base import SUL

class RNNSul(SUL):
    def __init__(self, rnn):
        super().__init__()
        self.rnn = rnn
        # keep track of the current hidden state
        self.hs = None

    # reset hidden state of an RNN to an initial hidden state
    def pre(self):
        self.hs = self.rnn.reset_hidden_state()

    # no other resetting is needed
    def post(self):
        pass

    def step(self, letter):
        # one hot encode element of input alphabet
        one_hot_encoded = self.rnn.one_hot_encode(letter)
        # get current output and update the hidden state
        out, self.hs = self.rnn.step(one_hot_encoded, self.hs)
        return out
```

5.2.2 Equivalence Queries

Let us describe competing approaches and how they differ from our extraction approach. Firstly, let us consider the similarities between approaches: all approaches use the L^* learning algorithm, and only differ in the choice of an equivalence oracle. The approach proposed by Weiss et al. [271] uses a white-box equivalence oracle, where white-box refers to the fact that they analyze the internal structure of the RNN in the search for the counterexample. On the other hand, Mayer and Yovine [145] approach an equivalence query from a black-box perspective by using a PAC adaptation of a random word oracle. We postulate that the most accurate approach for the extraction of input-output models from RNNs trained on regular languages needs to consider the hypothesis structure in the search for the counterexample. Therefore we propose the usage of model-guided equivalence oracles for the tasks at hand. Please note that all approaches use standard L^* implementations, and while their implementation might slightly differ, all implementations will return the model that conforms to the ground truth if all counterexamples are found.

White-box refinement-based equivalence oracle. This oracle was proposed by Weiss et al. [271]. Their equivalence oracle keeps track of two hypotheses: one hypothesis H_1 is learned using Angluin's L^* and the other hypothesis H_2 is a finite abstraction of the RNN obtained by partitioning the RNN's state space, thus they follow a white-box approach. That is, the approach is based on knowledge about the internal structure of the RNN under consideration. The intuition behind the oracle is that H_1 and H_2 need to be equivalent in order for them to be equivalent to the RNN. Whenever H_1 and H_2 disagree on a sample s, the true classification label of s is queried from the RNN. If H_1 incorrectly classifies s, then s is returned as a counterexample to L^*. Otherwise, the partitioning used to construct H_2 is refined. While this approach can benefit from additional information encoded in the RNN's state space, it may also suffer from exactly that. Improper partitioning of the high-dimensional state space may cause counterexamples to be left undetected.

PAC Oracle. A Random Word equivalence oracle with PAC guarantees was used by Mayer and Yovine [145] in their work on black-box model extraction from RNNs. The implementation conforms to the descrip-

tion of the random word PAC oracle outlined in Sect. 3.6. In the PAC framework [252], the number of tests performed by a random-sampling-based equivalence query can be configured to ensure that the returned hypothesis is an ϵ-approximation of the correct hypothesis with a probability of at least $1-\delta$ [156]. Here, an $\epsilon > 0$ bounds the generalization error, the expected number of examples incorrectly classified by the final hypothesis with respect to the sampling distribution D. To achieve such PAC guarantees, the number of tests performed by the lr^{th} equivalence query should be chosen according to

$$m_t = \frac{1}{\epsilon}(\log_e(\frac{1}{\delta}) + lr \log_e(2)) \text{ [156].}$$

Note that the PAC guarantees are relative to D, which may not reflect all interaction patterns with the SUL. Hence, a PAC learned model may have an error rate higher than ϵ when used under interaction patterns that differ from D. Consequently, such models may be quickly falsified with other testing methods, as shown in Sect. 5.3.

Fixing a sampling distribution enables probabilistic guarantees, but ignores learned information that is available in the form of the intermediate hypotheses. This may cause improper sampling of certain parts of a SUL, leading to premature halting of the learning procedure. Such issues are most prominent in non-strongly connected automata, automata with sink states, and combination lock automata. Nonetheless, for many systems, random testing with or without PAC guarantees has proved efficient [233].

Model-guided Equivalence Oracles. We use a selection of model-guided oracles described in Sect. 2.2.2 and Sect. 3.6. More concretely, we use the Random-W Method equivalence oracle (outlined in Alg. 2.3), as well as the transition focus oracle outlined later in this chapter. Both oracles exploit the structure of the current hypothesis in the search for the counterexample. These kinds of oracles do not provide PAC guarantees but usually ensure state or transition coverage over the learned model.

5.2.3 Parametarization of the Learning Algorithm

The proposed method does not limit the selection of the automata learning algorithm. A user could select any automata learning algorithm that can learn a deterministic SUL, such as L^* [14], *KV* [116], TTT [109], observation pack [99], or $L^\#$ [249]. We use L^* in our evaluation to enable fair comparison with competing approaches, while we used both L^* and *KV* in the scope of the TAYSIR competition. While these algorithms could be parameterized in various ways, most notably with different counterexample processing strategies, we put special emphasis on the early stopping.

Stopping. Until now we implicitly assumed that the SUL's behavior can be described by a regular language. However, RNNs are theoretically Turing-complete [223]. Even the RNNs trained to recognize regular languages might learn non-regular representations, that is, they could make generalization errors when compared to the ground truth. If we try to learn a DFA from an RNN modeling a non-regular language, learning may not halt, as equivalence queries may find infinitely many counterexamples. To counter such issues, we need to extend learning with a secondary stopping criterion in addition to positive results from equivalence queries. By stopping after a maximum number of learning rounds or upon reaching a maximum number of hypothesis states, we can learn automata accepting a regular approximation of a non-regular language modeled by an RNN. A similar approach has been coined *bounded* L^* [145]. Alternatively, we may limit the scope of the equivalence oracle to specific parts of the input space. This solution is appealing in a verification context. For example, a regular language describing a safety property could be used to learn a safe subset of an RNN's behavior. In Sect. 5.3.2, we present experiments with a context-free grammar, where we limit the scope of the equivalence queries by bounding the number of recursive rule expansions.

5.2.4 Research Questions

In our experiments, we will analyze model-guided conformance testing, learning in the PAC framework [145], and learning with the refinement-based oracle proposed by Weiss et al. [271]. We use the size of learned models to measure which approach produces the model closest to the true model capturing the input-output behavior of the RNN. This is motivated by the fact that every counterexample detected by an equivalence query increases the hypothesis size. Hence, model size provides us with a measure of accuracy. Therefore our first research question is:

RQ1: Learning with which equivalence oracle creates the most accurate models?

Since different counterexamples may lead to different intermediate hypotheses, the size of learned models alone is not an ideal measure of the effectiveness of equivalence oracles. A potential issue is that two learning approaches may learn two models with x and y states, respectively. Unless the larger value of x and y is equal to the number of states of the true model underlying the RNN, there may be covered by one of the models but not by the other and vice versa. To compare equivalence oracles more directly, we take a model learned by one approach and try to find counterexamples with another approach. Put differently, we try to falsify a model that was deemed correct by a different approach. This leads us to the second research question:

RQ2: Can the most effective equivalence oracle according to RQ1 falsify models that were deemed correct by the other two equivalence-oracle implementations?

RQ2 essentially aims to strengthen the conclusions from RQ1. RQ1 compares learning where all approaches share the same initial starting conditions. In contrast, RQ2 focuses on the most effective approach when starting from the conditions (the final hypotheses) found at the termination of the less effective approaches. In cases of positive answers to RQ2, we will also examine the effort necessary to falsify a learned model. This can, for instance, be measured as the number of test cases required for falsification. Alternatively to checking the size of learned models, other approaches are possible too. For example, we could consider measures of the language accepted by learned models with respect to a ground truth that we know, like precision and recall [262]. Since we solely compare $L*$-based approaches and due to the properties of L^*, we opted to consider the number of states as an objective criterion.

5.3 Experiments on Learning Automata from RNNs

We experimentally evaluated active automata learning instantiated with three different equivalence queries. We examine (1) our approach of model-guided conformance testing, (2) a white-box refinement-based equivalence oracle by Weiss et al. [271], and (3) a black-box approach providing PAC guarantees similar to the one proposed by Mayr and Yovine [145], who additionally introduced bounds. In the remainder of this section, we will dub the latter approach as PAC sampling-based learning.

Benchmarks. Tomita grammars [245] and the balanced parentheses grammar [271] are common benchmarks in automata learning and RNN-related domains [122, 145, 184, 265, 271]. We will use them to evaluate model-learning performance. We compare proposed methods on relatively small ground truth models for two reasons: firstly, while RNNs can theoretically be trained on all regular languages [153], accurately training them on bigger regular languages is hard in practice, and secondly, while the considered models might be small, we will show that learned RNN representations are much larger due to RNN generalization errors.

RNN Training Setup. We used a consistent approach for training all RNNs. That is, while the type (LSTM, GRU) or size (number of hidden layers, size of each layer) might vary, the other parameters of the training setup were consistent for all our experiments. For obtaining a training data set, we randomly generated sequences of various length using an FSM representing the considered language. Accepted sequences (i.e., words in the language) were labeled *True*, while sequences not in this language were

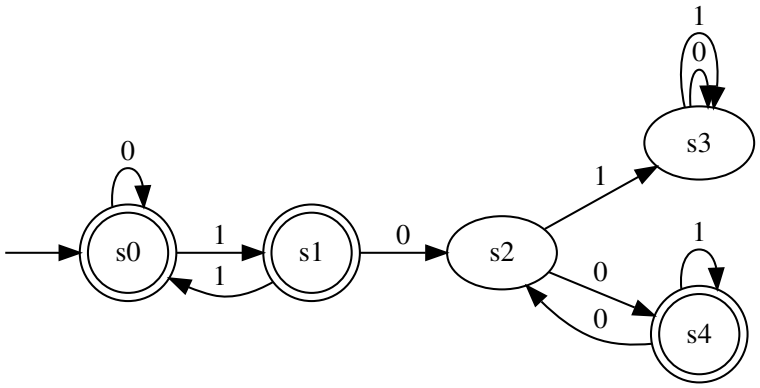


Figure 5.2: DFA representation of the Tomita 3 grammar

labeled *False*. Training and validation data was split in an 80:20 ratio. All RNNs were trained to 100% accuracy on the training and validation data sets.

Implementation. To enable a fair comparison with competing approaches, we used DyNet's [180] RNN implementations. We implemented our model-extraction framework[3] in AALPY. For the direct comparison with refinement-based learning we used the authors' original implementation[4]. All experiments were performed on a Dell Lattitude 5410 with an Intel Core i7-10610U processor, 8 GB of RAM running Windows 10 and using Python 3.6. All experiments were conducted on both types of supported RNNs (LSTM and GRU). Experiments were repeated multiple times to account for randomness found in the training and learning process. For each experiment a representative example was chosen such that it approximately corresponds to the average observed case.

5.3.1 Learning Models of RNNs Trained on Tomita Grammars

Before examining RQ1, let us illustrate learning models from RNNs for the example of the Tomita 3 grammar (shown in Figure 5.2). This five-state DFA was extracted and returned as a final hypothesis by multiple white-box [122, 271] and black-box [145] approaches. Furthermore, the learned automaton is indeed the same as the DFA of the Tomita 3 grammar itself, which *would* indicate that the behavior of the RNN conforms to the ground truth.

Conformance testing guided by hypotheses, i.e., guided by the model shown in Figure 5.2, enables a deeper investigation of the RNN's behavior. While other approaches were not able to find any further counterexamples and concluded the learning process with reporting said DFA, by testing with the random-W method, we were able to find additional counterexamples. It is important to note that such counterexamples thus reveal a faulty generalization by an RNN, i.e. they are adversarial inputs, since the DFA shown in the Figure 5.2 is equivalent to that of the ground-truth model used on which an RNN was trained on. By identifying such counterexamples, the model-guided conformance testing essentially falsifies the hypotheses returned by other approaches, such that a more detailed automaton can be learned. We observed that once a fault in the RNN generalization is found (i.e., an adversarial input), the size of the learned automata increases substantially (the next two hypotheses have 60 and 350 states). The effectiveness of model-guided testing stems from the fact that it uses information gained throughout the automata-learning process. In contrast to random sampling, it revisits identified states and explores new paths from there to find new states.

Table 5.1 shows the results of our experiments performed towards answering RQ1. In the experiments, we compare all three learning approaches based on the size of learned models, which provides us with a measure of accuracy. In the table, we report the number of counterexamples isolated during

[3] https://github.com/DES-Lab/Extracting-FSM-From-RNNs
[4] https://github.com/tech-srl/lstar_extraction

Table 5.1: Comparison of refinement-based [271], PAC sampling-based [145] and model-guided learning. All RNNs have 2 hidden layers with 50 nodes and were trained till 100% train and test set accuracy.

	Refinement-based Learning		PAC Sampling-based Learning		Model-guided Learning	
Tomita Grammar	Cex. Found	Model Size	Cex. Found	Model Size	Cex. Found	Model Size
1	1	2	4	25	11	46
2	2	3	3	8	11	44
3	3	5	3	5	85	3945
4	1	4	1	4	1	4
5	3	5	6	32	115	4953
6	67	11975	6	318	121	15084
7	1	2	1	5	7	33

the learning process as well as the final hypothesis' size. When conducting the model extraction experiments, we set a time limit of 500 seconds. Without such a time limit the learning process might not terminate if the RNN behavior cannot be captured by a regular language (see Section 5.2).

From the results we can conclude that model-guided learning creates larger automata than both refinement-based and PAC sampling-based learning, which can be explained by this approach finding more counterexamples. Recall that learned automata are minimal automata that are consistent with all queried information including all counterexamples. This means that the conformance-testing equivalence query finds more information and covers more of an RNN's input space. In fact, most models obtained via refinement-based learning conform to the ground truth models used for data generation. With PAC sampling-based learning and model-guided conformance testing, we were able to learn larger models. Hence, by using the black-box approaches we detected generalization issues that resulted in larger models that approximate the true input-output behavior underlying the examined RNNs more precisely.

For all seven Tomita grammars, the RNNs generalized very accurately within the length bound of the training set. This in turn caused refinement-based learning to learn models that conform to the models used for generating the training data. It did so by finding all counterexamples that are not adversarial inputs. Such non-adversarial inputs are words misclassified by intermediate hypothesis automata, but classified correctly by the RNN. Additionally, the technique managed to find adversarial inputs only for the RNN trained on the Tomita 6 grammar. By performing extensive model-guided testing, we did not only find all non-adversarial counterexamples, but we also found adversarial inputs for all Tomita grammars except for Tomita 4. Finding all non-adversarial counterexamples means that we are able to learn an intermediate hypothesis that is equivalent to the ground truth, whereas adversarial inputs allow us to detect discrepancies between the RNN under considerations and the ground truth.

PAC sampling-based learning managed to find adversarial inputs for some RNNs, while for others it found only counterexamples that led to learning of the model that corresponds to the ground truth model. From the minimality of models learned with L^*, we can conclude that the models created by refinement-based learning and PAC sampling-based learning are incorrect, except for the Tomita 4 grammar. Note that they are incorrect in the setting of exact learning, while models created by PAC learning are *probably approximately correct*.

Based on these results, we can answer RQ1. Learning with model-guided conformance testing created the largest models and therefore the most accurate models. This means that the learned models are the closest to the RNNs from which they were learned.

Apart from the findings concerning the accuracy of learned finite state models, we discovered that generalization issues and overfitting impacts the automata interpretation of the language learned by RNNs. Our evaluation shows that even RNNs trained from a simple ground truth with only two states (Tomita 1), can lead to a finite state model with at least 46 states —due to generalization issues and

Algorithm 5.1 Transition-Focus Equivalence Oracle

Input: number of tests n, test length L, same state probability p, hypothesis hyp, SUL
Output: counterexample or $\emptyset$

for $i \leftarrow 1$ **to** n **do**
  $testCase \leftarrow \epsilon$ ▷ Initialize test case to empty sequence ϵ
  $state \leftarrow reset(SUL, hyp)$
  for $j \leftarrow 1$ **to** L **do**
    if $random \in [0, 1] < p$ **then**
      $input \leftarrow choose(sameStateTransitions(state))$
    else
      $input \leftarrow choose(diffStateTransitions(state))$
    $testCase \leftarrow testCase \cdot input$
    $hypState = step(hyp, input)$
    $sulState = step(SUL, input)$
    if $hypState.output \neq sulState.output$ **then**

      return $testCase$
    $state \leftarrow hypState$

return $\emptyset$

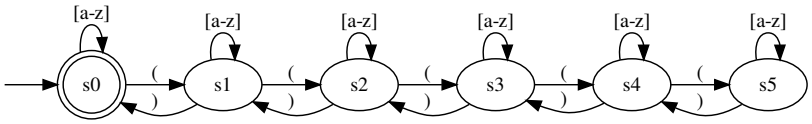


Figure 5.3: DFA of the balanced parentheses grammar bound to the depth 5.

overfitting. We generally observed that for well-trained networks finding counterexamples that are not adversarial inputs is less resource-intensive than finding generalization errors. To put this into perspective, it took usually several hundred test cases to find non-adversarial counterexamples, whereas the detection of the first adversarial input required up to 7000 test cases. It is important to note that finding subsequent adversarial inputs requires significantly fewer testing resources. This finding is explained in more detail in Section 5.3.2.

5.3.2 Learning Models of RNNs Trained on Balanced Parentheses

Balanced parentheses is a context-free grammar that accepts sequences of characters where all opening parentheses have matching closing parentheses. Any or no characters can be found in between parentheses. As balanced parentheses form a context-free language, we create a regular subset from it and use it for data generation. E.g., the bounded parentheses model limited to a depth of 5 nested parentheses is shown in Figure 5.3. We repeated the experiments on models with increasing depth of nested parentheses. Training and learning results were consistent irrespective of the balanced parentheses' nested depth. That is, the language learned by the RNN stayed mostly the same regardless of the depth that we tested.

We use experiments with this grammar as a basis for answering RQ2. For this purpose, we directly compare model-guided conformance testing, which produced the most accurate models, with the refinement-based and the PAC sampling-based learning. In contrast to Section 5.3.1, where we compared the entire extraction processes, we now focus on the falsification of the models obtained with the latter two approaches. That is, we applied refinement-based and PAC sampling-based learning to learn an automaton and use it as a basis for model-guided conformance testing.

Table 5.2: Counterexamples obtained during the refinement-based [271] learning process and counterexamples falsifying the learned model via model-guided conformance testing

Refinement-based Eq. Oracle		Random-W Eq. Oracle		Transition-focus Eq. Oracle	
Counterexample	Time (s)	Counterexample	Time (s)	Counterexample	Time (s)
))	0.34	((ik()fa))	4.6	((((()((i)y)t))()))	0.01
(())	0.16	((xvplzcmqzry()esp)sj)	4.3	(()(()()))()pu)	0.02
-	-	((dkulbgh(ajczy)o)lax)	0.39	xs(()(()))b)	0.01
		((i(h)dcaqgi)silnfg)	4.77	(y)k()(((()v)m)	0.01
		(((uuldz)t)zc)	3.83	(cr(s)()(cu()))(())h)	0.03
		(((fvtdjb)oeive)e)	1.16	a((j)))e)(((	0.02

In addition to RQ2, we want to briefly highlight how to integrate domain knowledge into the model-guided testing process. Knowledge about the structure of the balanced parentheses allows to develop a custom model-guided equivalence oracle especially suited for, but not limited to, this example. The *transition-focus* equivalence oracle is a model-guided equivalence oracle shown in Algorithm 5.1. The parameter p defines the focus of the test-case generation. E.g., a p value of 0.2 states that a newly chosen input will lead to a new hypothesis state 80% of the time and 20% of the time it will lead to the current state, i.e., correspond to a self loop in the hypothesis model. This equivalence oracle is especially suited for balanced parentheses, because only parentheses define transitions between states, whereas other inputs should lead to the same state. The transition-focus equivalence oracle generates test cases that explore the hypothesis structure primarily focusing on transitions between states. Compared to the random-W method, Algorithm 5.1 increases the probability of detecting a counterexample and potentially even an adversarial input.

Comparison with Refinement-Based Learning.

Table 5.2 shows the counterexamples obtained during the complete refinement-based learning process and counterexamples found with two model-guided oracles that the white-box technique did not find. The random-W method was able to falsify the learned model and transition-focused testing increased the efficiency of falsification. This demonstrates that model-guided conformance testing can find additional counterexamples that falsify models created with refinement-based learning, leading to more precise model learning. Hence, we can provide a first positive answer to RQ2.

Furthermore, these experiments show that model-guided testing improves accuracy also for non-regular languages and domain knowledge further improves efficiency. The transition-focus oracle improved the efficiency of falsification significantly. The found counterexamples are often adversarial inputs revealing the RNN's generalization faults. This shows that model-guided testing can be used as a tool for finding adversarial inputs, even with incomplete models.

Comparison with PAC Learning.

Now, we address RQ2 with respect to learning in the PAC framework, which has been used for analyzing RNNs by Mayr and Yovine [145]. We compare model-guided conformance testing and PAC sampling-based learning, by using the former approach to falsify a model learned by the latter approach. Finally, we will conclude with a discussion of PAC guarantees.

First, let us recap PAC-learning of automata. Equivalence queries are simulated through membership queries on words sampled according to a distribution D. By performing m_t membership queries for the t^{th} equivalence query we learn a model with a generalization error of at most ϵ with a probability of at least $1 - \delta$ [156]. This requires the sampled words to be independently and identically distributed according to the *fixed* distribution D. As a result, testing guided by hypotheses does not directly permit PAC guarantees, since the distribution with which words are sampled changes throughout learning. For

Table 5.3: Counterexamples obtained during learning with the PAC sampling-based oracle and counterexamples falsifying the learned model via model-guided testing

PAC Random Word Eq. Oracle		Random-W Eq. Oracle		Transition-focus Eq. Oracle	
Counterexample	# Tests	Counterexample	# Tests	Counterexample	# Tests
dnhugps)bch)	8	((gzbmcjin()weu))	15	((()p))	1
om(a(jvu))	2163	((qcss(sizx)uevu))	173	k(()(x(xv)))	4
-	-	((u()tysjxu))	157	(zz((t)))	14
		((isr(t)u))	354	((x(()(gh)())z))	5
		(((wdwv)onvu))	234	(t(v()))	9
		((aao()kgkfk))	61	((j()q))	11

Table 5.4: Learning processes of PAC sampling-based and model-guided learning of an RNN trained on Tomita 3 grammar. The size of the current hypothesis and number of tests needed to falsify it are given for each round

	Learning Round	1	2	3	4	5	6	7	8	9	10
PAC sampling-based learning	Hypothesis Size	1	4	5	21	25	46	-			
	# Tests to Falsify	10	3	33929	20143	30111	-				
Model-guided learning	Hypothesis Size	1	4	5	26	39	257	291	316	346	431
	# Tests to Falsify	4	5	1428	11	124	12	4	9	13	-

learning in the PAC framework, we fix a distribution for sampling at the beginning of learning. We sample input symbols according to a uniform distribution over the inputs and we choose the length of each test case according to a uniform distribution over the integer interval $[4..20]$. In every equivalence query, we perform m_t random tests with $\epsilon = 0.001$ and $\delta = 0.01$. Values of ϵ and δ were selected to ensure high PAC guarantees, providing the PAC sampling-based oracle with a high amount of testing resources. The concrete number of queries is calculated with the equation presented in Section 5.2.2. Note that we examine the number of test cases needed to find a counterexample, which may be lower than m_t, the maximum number of test cases executed by an equivalence query.

To answer RQ2, we perform the same experiments as above for refinement-based learning. We learn a model with the PAC sampling-based oracle and try to falsify it with model-guided testing. Table 5.3 shows results from these experiments. We can see that the PAC sampling-based learning managed to find only two counterexamples. This finding is consistent with the experiments presented by [271]. Even with the imprecise model learned with the PAC sampling-based oracle that found only two counterexamples, model-guided testing was able to quickly falsify said model. Hence, we can answer RQ2 positively. In Table 5.3, we can also see that it took a low number of test cases. It took on average only about 166 test cases to find additional counterexamples with the random-W method.

5.3.3 Analyzing RQ2 on the Tomita 3 Grammar

In further experiments, we compared model-guided learning with PAC-based learning when learning models from an RNN trained on the Tomita 3 grammar. Table 5.4 shows the results from these experiments. These experiments further demonstrate the advantage of exploiting the structure of the learned model during test-case generation. For both approaches, we observe that finding the first two counterexamples was equally fast, as those are the counterexamples leading to the creation of a model conforming to the ground truth. From that point onward, we notice significant differences between the two approaches. To falsify the ground truth model (learning round three) a substantial increase in testing resources is needed. We can observe that the process of finding subsequent counterexamples (adversarial

inputs) by the PAC sampling oracle required significantly more test cases than its model-guided counterpart. Furthermore, the PAC sampling-based oracle was unable to find all counterexamples, terminating the learning in round 6 and returning a 46-state model. An interesting observation can be made concerning the number of tests needed to falsify the hypothesis in the case of model-guided testing. To falsify the ground truth model, the random-W method required 1428 tests, while all subsequent equivalence queries returned a counterexample with a low amount of testing. This is due to the fact that once the counterexample falsifying the ground truth model is found, it is processed, which leads to an increase of the hypothesis size. This increased hypothesis encodes more knowledge about the structure of the system under learning, thus making the search for counterexamples easier. This observation confirms the need for exploiting the model structure in the test-case generation.

Finally, let us examine the guarantees provided by PAC learning. Note the meaning of the error-rate parameter ϵ in the PAC-based setting: random testing may also be able to falsify a learned model, but it would require on average at least $\frac{1}{\epsilon}$ tests. We see in Table 5.3 that model-guided testing requires between 11 and 354 tests for falsification, which is consistently lower than $\frac{1}{\epsilon} = 1000$. Hence, it tests the SUL more efficiently. Although PAC results in automata learning provide a quantification of correctness, such a quantification may result in a false sense of security. If the sampling distribution for simulating equivalence queries does not match usage patterns, PAC guarantees are not very useful. Hence, if expected usage patterns are not known, model-guided testing is preferable. This approach creates diverse data and it benefits from the knowledge gained during learning.

5.4 Application of the Proposed Method in the TAYSIR Competition

TAYSIR competition [69] was a competition held in the scope of International Conference on Grammar Inference (ICGI) 2023. The goal of the competition was the development of methods that yield simple and interpretable models from pre-trained RNNs and transformers. The competition had 40 registered teams, among which seven of them submitted their solutions.

Competition Structure. The competition consisted of two tracks: the binary classification track and the language modeling track. In the first track, RNNs were trained on datasets whose labels were in the Boolean domain, while in the second track networks were trained to predict a probability distribution over the set of potential next symbols given any prefix. Both tracks considered networks trained on a discrete input domain. Participants of the competition were given pre-trained RNNs and transformers that were trained on unknown tasks, as well as the validation set used during the network training. The goal of the competition was the extraction of "surrogate models", that is, models that faithfully capture the input-output behaviors of the pre-trained networks. Provided solutions were judged for their accuracy and simplicity. The accuracy of submitted solutions was evaluated on another dataset unknown to the participants. The accuracy metric for Track 1 was based on the misclassification rate between the "surrogate model" predictions and the ground truth, while the accuracy of surrogate models for Track 2 was judged with an mean squared error (MSE). Since participants could use various approaches, the simplicity of the solution was judged as the combination of the total runtime and memory usage used during the prediction phase.

Method Overview. We used the extraction method proposed in this chapter to mine models of pre-trained RNNs and transformers. As done throughout this chapter, we considered RNNs and transformers as a black box and learned a minimal automaton representing their input-output behavior. Our method was suitable for Track 1 without any modifications as both input and output alphabets were discrete, while for Track 2 we added an output mapper that maps the floating-point output of a neural network to a moderately-sized finite discrete set of outputs. The whole codebase required to extract models from RNNs, as well as learned models, can be found in the public repository [163].

Model Memory Footprint and Execution Time. All models in Track 1 are encoded as DFAs, while models in Track 2 are encoded as Moore machines. All models are saved as a Python dictionary, which defines the output of each state and all outgoing transitions. Therefore, models have a small memory footprint. For example, a 1000-state DFA with an input alphabet of size 10 only uses 37kB. The execution time and memory footprint of these models is minimal, as we only keep track of the current state while iterating over a test sequence.

Precomputation of the Validation Set. For both tracks, we saved output values for all sequences of the given validation set. This was done to speed up the testing of the extraction procedure. This step is completely optional, but we have included it in the repository to speed up the reproduction of results. We also used these sequences as additional test cases in equivalence queries.

5.4.1 Track 1: Binary Classification

In Track 1, we used the method described in this chapter without any modifications. Our method is suited for the extraction of regular languages from RNNs, and since we consider the SUL as a black box, the adaptation to a transformer architecture was trivial. However, not all networks in the competition were trained to recognize a regular language. Even if they were trained on a regular language, the RNN's input-output behavior might not be regular due to faults in the RNN's generalization compared with the ground truth. In these scenarios, our method will learn a regular approximation of a non-regular language. These approximations could conform to the SUL up to the specific sequence length, e.g., we might learn a regular language that models the SUL's behavior for all sequences of maximum length 10. However, we put no such restrictions. Note that for some experiments we used *KV* instead of L^*, as early stopping is more easily controlled in *KV*[5]. This allowed us to provide an upper limit to regular approximations of non-regular languages.

We outline some findings and specifics of the extraction procedures:

- **Datasets 2,3,4,5,7**: All networks behave as regular languages, with minimal DFA sizes of 8, 9, 5, 5, and 2. Interestingly, the extracted model trained on Dataset 2 that achieves 100% accuracy has 8 states, but when a stronger testing oracle is used we can also extract a model with 10 states.
- **Dataset 6:**: The network trained with Dataset 6 also behaves as a regular language, with 18 states being found without a need for strong testing. These 18 states achieve a 0.0002% error rate on the unknown validation set. However, a strong equivalence oracle can find many counterexamples to this 18-state model, leading to a substantial increase in hypothesis size. We postulate that the underlying RNN was trained on a regular language, but it fails to generalize to all sequences with respect to the ground truth model.
- **Dataset 1**: The network trained with Dataset 1 potentially behaves as a context-free language, as we noticed that a 200-states DFA achieved 0.12% error rate on the unknown validation set, while a 7708-state DFA achieved 0.075% error rate. The 7708-state DFA conforms to all sequences in the known validation set.
- **Dataset 8:** We were unable to identify any meaningful structure in this network. Automata learning would run into a practically infinite loop and early stopping of that loop at various points yielded models that had no improvement in accuracy.
- **Datasets 9, 10, 11:** The behavior of these networks was not regular, and learning was bounded to 200, 1500, and 500 learning rounds with the *KV* algorithm. The learned models (regular approximations of a non-regular language) have low error rates (0.007%, 0.014%, 0.007%). This shows that automata learning can learn relatively accurate regular approximations of non-regular languages.

[5]As we know that an n-state DFA will require $n - 1$ learning rounds.

Table 5.5: Parameterization and results of model extraction for Track 2.

Dataset	1	2	3	4	5	6	7	8	9	10
Error Rate (MSE $\times 10^6$)	0.175	0.0097	3×10^{-5}	6×10^{-6}	7×10^{-8}	0.1971	0	0.0443	0	0.1237
Model Size	866	131	110	105	123	318	170	162	55	1412
Learning Rounds	500	100	50	50	50	200	100	100	30	200
# Intervals	200	10	10	10	12	20	15	15	10	20

5.4.2 Track 2: Density Estimation

In Track 2, we have used the same technique as in Track 1, but with the addition of an output mapper. Once a concrete output value is obtained after executing an input sequence, the applied output mapper maps a virtually unbounded set of output values in the range [0,1] to the final discrete set of abstract outputs.

Mapper Generation. To create a mapper, we start by computing and sorting the set of all observed outputs for known validation sequences. This sorted list of outputs is then partitioned into a predefined number of intervals. Each interval acts as a discrete output which is used to represent all values that fall in this interval. This interval partitioning creates more intervals in "more frequent areas" of the output domain. This method is also known as *equal frequency intervals discretization* [65].

For example, consider that the validation set consists of nine input sequences, and the sorted obtained outputs are: [0.01, 0.015, 0.02, 0.4, 0.45, 0.5, 0.7, 0.85, 0.95]. Let us set the number of intervals to three, therefore dividing the sorted output list into three equally sized sublists. We then map discrete interval identifiers to the mean of each sublist. We end up with the following mapping: $\{$b1: 0.105, b2: 0.45, b3: 0.83$\}$. These mean values are used to compute the abstract value of a concrete output by choosing the interval with minimal distance to the concrete output. Suppose we are then executing an input sequence on a network and obtain the following outputs: [0.01, 0.3, 0.5, 0.99, 0.2]. This output sequence would be abstracted to [b1, b2, b2, b3, b1].

On the Number of Intervals. For all experiments, the selected number of intervals can be seen in Table 5.5. A higher number of intervals will yield more accurate models, but with higher automata learning costs to account for a larger output alphabet. Therefore, we set a relatively low number of intervals and focus on the quality of automata learning, as we postulate that hitting the "correct" interval, i.e., predicting the correct interval with a learned model, is more important than having more intervals. More details on the learning parameterization can be found in the linked repository.

Learning outcome. Given the discrete input alphabet and a discretized output alphabet computed by a mapper, the learning algorithm (L^* or *KV*) infers a Moore machine that approximates the SUL's behavior. The output of every state in the learned Moore machine is an interval identifier (eg. b1, b2,..). The extracted model makes predictions by simply tracing an input sequence through the model, and returning the mean value of the reached interval.

Results. Results (error rate and number of states in the learned model) and learning parameterization (number of learning rounds and number of intervals) are shown in Table 5.5. All extracted models achieve a low mean-squared error rate ($\leq 1 \times 10^{-6}$). These results indicate that models learned with our output mapper could serve as sufficiently accurate "surrogate models" for networks trained on language modeling tasks.

5.5 Concluding Remarks

Combining formal methods and machine learning, in this particular case, automata learning and RNNs provides us with a better understanding of an RNN's decision-making process. Learned models of RNN input-output behavior could help researchers in the analysis of factors that influence RNN training and generalization capabilities, e.g., the properties of the training data could be compared with the learned model to better understand which properties of the data contribute to better RNN training.

While the proposed method can be used to extract input-output behavioral models from RNNs trained on regular languages, it still cannot cope with the high-dimensional nature of data on which RNNs are usually trained. To tackle this challenge, researchers have proposed looking into the RNNs for a solution. In the next chapter, we thoroughly analyze the long-standing hypothesis that an RNN's hidden-state vectors form semantically meaningful clusters.

RQ 1.1 Can automata learning be used to efficiently extract behavioral models from RNNs?

We showed how automata learning can be used to extract input-output behavioral models from RNNs trained on formal languages. In our evaluation, we showed that the proposed method greatly increases the accuracy of the learned model and is able to efficiently falsify models returned by competing methods. The validity of our method was further validated by its successful application in the scope of the TAYSIR competition, where it outperformed all competing approaches. Work presented in this chapter further strengthens points made in **RQ1**, as we have demonstrated AALPY's capabilities in the machine learning domain.

RQ 2.2 What is an appropriate abstraction for the modeling of RNNs?

In the scope of Track 2 of the TAYSIR competition we observed that equal frequency intervals discretization can be used as an abstraction technique for RNNs trained on language modeling tasks, that is, on the RNNs that whose output domain is in $\mathbb{R}$.

6

On the Relationship Between Semantic Structures and RNN Hidden-State Vectors

Declaration of Sources

The work presented in this chapter is based on our work "On the Relationship Between RNN Hidden-State Vectors and Semantic Structures" [172], which was accepted at Findings of ACL 2024.

6.1 Motivation

In the previous chapter, we have shown how automata learning can be used to learn input-output behavioral models of RNNs. For that purpose, we trained RNNs on regular languages that have provided us with a formal ground truth against which we could assess RNN generalization capabilities. In this chapter we take a different view: while in the previous chapter we analyzed RNN input-output behavior from the black-box perspective, in this chapter we turn to the internals of RNNs in hopes of better understanding the RNN decision-making process.

An important, yet scarcely examined, hypothesis in the field of formal RNN analysis postulates that the hidden state vectors visited by an RNN while processing data form clusters of semantically similar states. Consider sentiment analysis for movie reviews as an example [64]. The sentences *"This movie is great."* and *"This movie is awesome."* are semantically similar, but syntactically different. An RNN processing these sentences is assumed to traverse states that belong to the same state clusters.

Hence, the "*clustering hypothesis*" can provide powerful analysis capabilities. It facilitates the creation of equivalence classes in the internal state space of RNNs by clustering. Such a discretization of high-dimensional real-valued RNN state vectors to a finite number of equivalence classes enables the application of a plethora of analysis tools. These classes enable further analyses, e.g., through the extraction of finite-state machines (FSMs) with states corresponding to clusters. Unlike RNNs, FSMs are well-understood and amenable to model-based reasoning.

In recent years, the clustering hypothesis was the basis of several well-received RNN analysis methods. Dong et al. [64] relied on the clustering hypothesis to construct finite-state models over observed clusters. Through probabilistic model checking, they identified adversarial data. Weiss et al. [271] developed an efficient technique for extracting deterministic finite automata from RNNs, which was extended

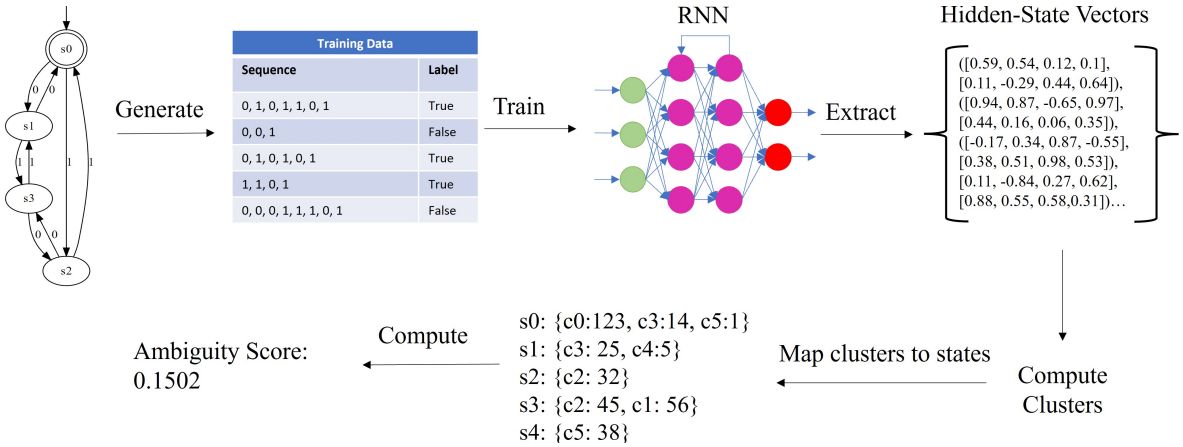


Figure 6.1: Overview of the process for assessing the quality of clustering functions

to context-free languages in subsequent work [279]. Clustering could also help to extract finite-state controllers from RNN policies of deep reinforcement learning agents, replacing other more specialized discretization methods [122].

While the aforementioned approaches [64, 271, 279] relied on some form of the clustering hypothesis, they also encountered, and subsequently coped with inaccuracies of computed clustering functions. Instead of using clusters as abstract states directly, Dong et al. [64] learned probabilistic automata over observed clusters. The applied learning algorithm may introduce multiple states for individual cluster observations, which reduces the error introduced by imprecise clustering. While successful applications of the clustering hypothesis *might imply* its soundness, the validity of the hypothesis still remains questioned. For example, already in early work Kolen [119] warned against methods that rely on state-space discretization (clustering) due to the inherent information loss. Also, Zeng and Smyth [283] noticed problems with the clustering approach.

6.1.1 Guiding Idea

Since validation and verification of RNNs should not be based on anecdotal evidence for the validity of the clustering hypothesis, we return to this open question and thoroughly study if the hypothesis holds. The results of our empirical study shall quantify the precision of clustering-based abstractions.

More precisely, first, we train RNNs to recognize formal languages with known and minimal ground-truth automaton representations. Then, we process data with the RNN to investigate the following research objectives: Are an RNN's hidden-state vectors linearly separable (RO1)? Is the clustering hypothesis assumed for RNNs valid (RO2)? Does the level of the Chomsky hierarchy affect clustering (RO3)? We approach these questions by simulating validation data on trained RNNs to extract visited hidden state vectors. At the same time, we track the unique automaton state corresponding to each RNN state. Then, we partition the sampled RNN states using multiple clustering functions and we train linear multiclass classifiers to classify RNN states to automaton states. The latter provides a linear separation of the state space. Unlike for clustering we use known ground-truth states, therefore it establishes an upper bound for the precision of clustering with linear cluster boundaries, like k-means, the most popular clustering technique. Hence, RO1 can be seen as a prerequisite for some clustering techniques. To answer RO2, we analyze the correlation between derived clusters and automata states and assign an ambiguity score to each clustering configuration. Ideally, each cluster (or a set of clusters) should correspond to a unique state in the respective ground-truth automaton. An overview of this process can be seen in Fig. 6.1. For the first two research objectives, we focus on regular languages. Regular languages provide a good basis for analysis, but many practical applications, such as natural language processing[1] require

[1]With the recent success of transformers one might have the impression that RNNs are obsolete. However, recent work [34] shows that it is too early to close this debate.

more expressiveness. Therefore, for RO3 we move one layer up in the Chomsky hierarchy and analyze the clustering hypothesis on context-free languages that can be modeled with deterministic pushdown automata..

Structure. This chapter is structured as follows: in Section 6.2 we extend the background knowledge required for our analysis. In Section 6.3 we formulate the research objectives and present techniques and accuracy metrics that we use in Section 6.4, where we perform the empirical evaluation of the clustering hypothesis. Finally, we conclude our analysis in Section 6.5.

6.2 Preliminaries

Throughout this chapter, we rely on standard definitions of DFAs, Moore machines, and RNNs outlined in Chapter 2. We further introduce additional notation over RNNs and formally define pushdown automata.

6.2.1 Pushdown Automata

Definition 6.1.
A deterministic pushdown automaton (PDA) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ *where:*

- Q *is a finite set of locations,*
- Σ *is a finite input alphabet,*
- Γ *is a finite stack alphabet,*
- $\delta : Q \times \Sigma \times \Gamma^* \to Q \times \Gamma^*$ *is the transition function,*
- $q_0 \in Q$ *is the initial location,*
- $Z \in \Gamma$ *is the initial stack symbol and*
- $F \subset Q$ *is the set of accepting locations.*

In contrast to DFAs, the state of a PDA is a pair $(q, \gamma) \in Q \times \Gamma^*$, where q is a location and γ is a stack of symbols from Γ. Analogous to DFAs, we extend δ to sequences w where $\delta(q_0, Z, w) = (q, \gamma)$ is the state reached after processing w. A PDA defines a context-free language L, where $w \in L$ iff $\delta(q_0, Z, w) = (q, \gamma)$, $q \in F$, and the stack γ is empty.

6.2.2 Functional View of RNNs

We view an RNN as a pair of functions (r, o), where $r : \mathbb{R}^h \times \mathbb{R}^m \to \mathbb{R}^h$ updates the hidden state based on the previous hidden state h_{t-1} and the current input i_t. We use a one-hot encoding, denoted with ρ, for discrete inputs from an alphabet of size m. For simplicity, we also use this view for LSTMs by analyzing the state space spanned by the concatenation of the hidden state h_t and the cell state c_t of LSTMs. The output function $o : \mathbb{R}^h \to C$ maps the current hidden state to an output class in C. We train RNNs on sequences sampled from languages defined by DFAs, Moore machines, and PDAs. For DFAs and PDAs, we have $C = \{true, false\}$, where $true$ denotes acceptance of the sequence processed so far. For Moore machines, we have $C = O$, where O is an output alphabet and $o \in O$ maps to the last output produced by the corresponding Moore machine.

6.2.3 Multiclass Classification

We examine the correspondence between hidden state vectors and ground truth models represented by deterministic finite automata. Given a finite sample of vectors in $\mathbb{R}^h$ labeled with corresponding automaton states from a set Q, we formulate a supervised multiclass classification problem. We apply (generalized)

linear models to solve the problem so that their accuracy provides insights about the (piecewise linear) separability of $\mathbb{R}^h$ into decision regions corresponding to Q.

In this setting, we have a sample from $\mathcal{S} \subset \mathbb{R}^h \times Q$. A pair $(h, q) \in \mathcal{S}$, contains a vector h from the hidden state space of an RNN and an automaton state q, which is the class label for the classification problem. Our goal is to find a function $cl : \mathbb{R}^h \to Q$ that classifies vectors correctly. We apply two representative techniques for classification: linear discriminant analysis (LDA) and logistic regression (LR). Both techniques yield linear models, meaning that their output classes depend linearly on the hidden state vectors serving as inputs. logistic regression (LR) is actually a generalized linear model since it uses a nonlinear transformation of the input space. For more information we refer to the literature [29].

6.2.4 Clustering

Clustering deals with the identification of structure in data, an important problem in unsupervised learning [141]. In contrast to multiclass classification, clustering techniques take unlabeled data points as inputs. Clustering groups data points into clusters, such that ideally data points in the same clusters are *similar* to each other and data points from different clusters shall be dissimilar from each other. The notion of similarity is generally a problem-specific measure. The input to a clustering technique is an unlabeled sample of data points $\mathcal{US}$, in our case: $\mathcal{US} \subset \mathbb{R}^h$. Thus, we can view a clustering as a function $c : \mathcal{US} \to K$ assigning cluster labels to data points. We focus on popular, efficient techniques available in the library scikit-learn [191]. We apply k-means [140], a partitional clustering technique [141] and three density-based clustering techniques: mean shift [54], DBSCAN [68], and OPTICS [16].

K-means partitions data into k clusters (where k is user defined), assigning each data point p to the cluster whose center is closest to p. The center is the mean of all data points in a cluster. K-means creates piecewise linear decision boundaries, where the distance from centers of a neighboring cluster is equal. Density-based techniques create clusters of arbitrary shape that are identified as regions with a high density of data points. **DBSCAN** defines regions as clusters if at least $minNeighbors$ points are in a neighborhood of a size defined by a radius ϵ. By merging clusters that are close to each other, it finds arbitrarily-shaped clusters. **OPTICS** follows the same basic approach as DBSCAN, but improves it by mitigating the sensitivity on the neighborhood size ϵ [274]. Like k-means, **mean shift** clusters a dataset based on centroids. Conversely, however, it attempts to find maxima in the density function underlying the distribution of the data and does not require a preset number of clusters k. It uses a bandwidth parameter bw to define regions for the update of centroids.

6.3 Analysis Method

This section presents the basis for the analysis of the "clustering hypothesis". Firstly, we formulate our research questions and introduce the setting. Then, we provide details on the analysis including accuracy measures.

6.3.1 Research Questions

Setting. In our experiments, we train RNNs on formal languages over an alphabet Σ. For each regular language $L \subseteq \Sigma^*$, we sample a dataset $D \subset \Sigma^* \times \mathit{label}$, where each element in D is a pair (w, label) such that *label* denotes weather w is in L. We split the sample D into training data D_t and validation data D_v of sizes n_t and n_v. The former is solely used for training whereas the latter provides a stopping criterion for training and data for the analysis of the clustering hypothesis. For the following discussion, we fix a language $L \subseteq \Sigma^*$ and a sample dataset $D \subset \Sigma^* \times \mathit{label}$.

For a **regular language** L, let $A = \langle Q, q_0, \delta, F \rangle$ be a minimal DFA accepting exactly L and let $R = (r, o)$ be an RNN over the hidden state space $H = \mathbb{R}^h$ trained to recognize L. By processing every word w in D_v simultaneously with the recurrent part r of R and the minimal DFA A, we determine

Algorithm 6.1 Labeling of hidden states with automaton states

Input: $A = \langle Q, q_0, \delta, F \rangle$, RNN (r, o), initial hidden state h_0, validation data $D_v \subset D$
Output: pairs of hidden and automaton states $\mathcal{HQ}$

```
1: HQ ← ∅
2: for all (w, label) in D_v do
3:     q ← q_0, hs ← h_0
4:     HQ ← HQ ∪ {(hs, q)}
5:     for all i in w do
6:         q ← δ(q, i)
7:         hs ← r(hs, ρ(i))
8:         HQ ← HQ ∪ {(hs, q)}
9:
10: return HQ
```

the hidden states traversed by R as well as the corresponding automaton states reached by A. We store these data as pairs $(h, q) \in H \times Q$. For the remainder of this section let $\mathcal{HQ} \subset H \times Q$ be a concrete sample of such pairs and let $\mathcal{H}$ be the same sample without states Q. Algorithm 6.1 formalizes the creation of $\mathcal{HQ}$. For a **context-free language**, we follow an analogous procedure backed by a PDA $A = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$. In this case, we store triples $(h, q, \gamma) \in H \times Q \times \Gamma^*$, i.e., we keep track of the configuration in addition to the automaton location.

RQ1: Are an RNN's hidden-state vectors linearly separable? As a first step, we investigate whether hidden-state vectors can be (piecewise linearly) separated into regions corresponding to automaton states with a focus on regular languages. We formulate this as a multiclass classification problem based on $\mathcal{HQ}$, the sampled hidden-state vectors labeled by automaton states. That is, we train a classification model on $\mathcal{HQ}$ to learn a function $cl : \mathbb{R}^h \to Q$ and we evaluate its accuracy at classifying hidden states correctly. We rely on (generalized) linear models LDA and LR for this task. This choice is motivated by the observation that non-linear functions were not required in a similar setting [152]. While *piecewise linear separability* and low classification error of linear models *do not imply that the hidden states form well-defined clusters*, we gain insights into the structure of the hidden-state space. Linear separability can be seen as a prerequisite for clustering with linear cluster boundaries, like k-means. Hence, by using linear models for separability we enable a fairer comparison to clustering.

RQ2: Is the clustering hypothesis assumed for RNNs valid? This research question concerns the validity of the clustering hypothesis for regular languages. To examine it experimentally, we consider two aspects related to the clustering of RNN states.

- **The definition of a cluster.** Works based on the hypothesis apply different clustering techniques, hence we will analyze the effect of different popular clustering techniques.
- **Usefulness of a cluster.** The appeal of clustering stems from its potential as an abstraction for RNNs, thus enabling model-based reasoning [64]. Hence, clusters should have similar qualities as abstractions for software systems. Concrete hidden states *abstracted to the same cluster* should behave similarly. Moreover, an abstraction should be small enough to enable efficient subsequent analyses, thus we will examine the number of detected clusters.

We will analyze these aspects based on experiments with regular languages, where we know the minimal ground-truth automaton. Hence, answering **RQ2** in this context can also be formulated as: **Do hidden state clusters correlate with states of an automaton accepting the same language?**

RQ3: Can the clustering hypothesis be extended to languages higher in the Chomsky hierarchy? While the clustering hypothesis was postulated for RNNs trained on regular languages, it has also been assumed for RNNs trained on natural language [64]. Since natural language cannot be formally modeled and analyzed in a similar manner as regular languages, we instead extend our analysis to the next level in the Chomsky hierarchy, i.e., to context-free languages. To formally analyze the clustering hypothesis in

such a setting, we extend **RQ1** and **RQ2** to RNNs trained to recognize a subset of context-free languages that we modeled with deterministic PDAs.

6.3.2 Accuracy Metrics

We discuss the evaluation of the accuracy of RNNs and clusterings below. The former is based on the misclassification of words w.r.t. the ground-truth language L. Clustering accuracy is based on the ambiguity resulting from interpreting clusters as states of a finite-state model, compared to the states of the ground-truth automaton A.

RNN Accuracy Validation. We validate the accuracy of an RNN $R = (r, o)$ by sampling words Σ^* to form an accuracy-validation set AV and checking whether R agrees with the ground-truth L, i.e., accepts the language L. For each word $w \in AV$ we process w with R by repeatedly applying r to get the final hidden state $h_{|w|}$ and computing the RNN output via $ro = o(h_{|w|})$. Then, we check whether ro agrees with the ground truth, where we define $agree_{R,L}(w) =_{def} ro \leftrightarrow w \in L$. We define the accuracy of R as

$$Acc_R(AV) = \frac{|\{agree_{R,L}(w) | w \in AV\}|}{|AV|}.$$

Measuring the Quality of Clustering. In order to provide an intuition for the optimality of clustering, let us focus on regular languages first. Observe that *all data points in a cluster should correspond to a unique state in the ground truth A*. Formally, we can view a clustering as a function $c : \mathcal{H} \to K$ mapping sampled hidden states to cluster labels. This lets us compare a clustering c to the optimal mapping $hq : H \to Q$ based on the samples in $\mathcal{HQ}$. To relate c and hq, we define the clustering of c as optimal if $hq = \alpha \circ c$ holds, where $\alpha : K \to Q$ renames cluster labels to automata states. Such a c allows extracting a DFA with states K from an RNN R that is equivalent to ground truth A if $Acc_R(AV) = 1$ and AV is large enough. To empirically evaluate a concrete c, we define the mismatch between c and hq based on entropy.

Before formally defining the mismatch between c and hq, which we term *ambiguity* of c, let us analyze criteria for good versus bad clusters. To apply clustering as an abstraction over H, a cluster should group semantically similar states. Conversely, semantically different states should not be in the same cluster. Hence, we can derive that a clustering with $|K| < |Q|$ cannot be optimal, since all states in a minimal DFA are different, i.e., distinguishable through future behavior. In other words, a DFA extracted from the corresponding RNN with states identified by clusters K would not be equivalent to the ground-truth DFA A. If $|K| \geq |Q|$, the clustering may have a small mismatch, but the RNN may have learned a non-minimal representation if $|K| > |Q|$. The latter would not be desired for abstraction due to efficiency reasons. Next, we consider the best- and worst-case distributions of states in a cluster. Ideally, all hidden states in a cluster should map to the same automaton state. The worst case is achieved by a uniform distribution over states, as then the cluster would possess no semantic relation to the concept that has been learned.

We define ambiguity via the well-known concept of *information entropy*, which measures the degree of uncertainty (mismatch) in our clustering mappings. The entropy of our best-case clustering would be 0, while the worst-case clustering (uniform distribution) would give maximal entropy. Given, the general definition of entropy $Entropy(X) =_{def} -\sum_{x \in \mathcal{X}} p(x) \log p(x)$ with X being a random variable with outcomes in alphabet $\mathcal{X}$ with distribution $p : \mathcal{X} \to [0, 1]$, we can instantiate it to measure the uncertainty in our cluster mappings via:

$$amb(k) = -\sum_{q \in Q} \frac{n_{q,k}}{n_k} \log_{|Q|} \frac{n_{q,k}}{n_k} \text{ where}$$

$$n_{q,k} = |\{(h, q) \mid (h, q) \in \mathcal{HQ}, c(h) = k\}| \text{ and } n_k = \sum_{q \in Q} n_{q,k}$$

$$amb(c) = \frac{\sum_{k \in K} amb(k)}{|K|} \qquad wamb(c) = \frac{\sum_{k \in K} amb(k) \cdot n_k}{|\mathcal{HQ}|}$$

By using the logarithm with base $|Q|$ we ensure that the ambiguity is normalized to the interval $[0, 1]$. For the ambiguity of a clustering function, we compute the average of all clusters and the weighted average $wamb$. We noted above that ideally $hq = \alpha \circ c$ for a renaming α. This is achieved iff $amb(c) = 0$. We say that clustering is perfect if it achieves a (weighted) ambiguity of zero.

For PDAs, we sample triples (h, q, γ), i.e., we need to match RNN states h to PDA states (q, γ). However, considering that clustering shall serve as an abstraction, we cannot take the complete stack γ into account. This would lead to a very large abstract state space, making linear separation and clustering of the RNN states superficially simple and the resulting abstraction not useful. As acceptance of words depends on whether a location q is accepting and the stack is empty, hidden states should form clusters corresponding to locations and stack size. In addition to stack size, the top element of the stack affects how PDAs process words. Therefore, we abstract away from the concrete stack configuration to either stack size or the top element. Altogether, we apply three abstractions in the experiments: the first stackful abstraction $abs_{tos}(q, \gamma) = (q, top(\gamma))$ maps the PDA states to the current location and top of the stack, the second stackful abstraction $abs_{sl}(q, \gamma) = (q, |\gamma|)$ maps PDA states to pairs of location and stack size, whereas the stackless abstraction $abs_l(q, \gamma) = q$ completely ignores the stack.

To enable a straightforward comparison between classification models and unsupervised clustering, we apply ambiguity also for classification models, by interpreting predicted classes as cluster labels. Note that an ambiguity of zero coincides with a misclassification rate of zero. Thus, zero ambiguity of linear models implies (piecewise linear) separability on the sample dataset $\mathcal{H}$.

Alternatively to our notion of ambiguity, we could also use normalized mutual information (NMI) [57], a commonly used estimate of clustering quality with an information-theoretic interpretation. However, clustering size affects NMI such that a perfect, but slightly non-minimal clustering may have a lower NMI than an ambiguous, but small clustering. Since the former (a perfect, non-minimal clustering) is likely more useful for RNN analyses than the latter (imperfect clusterings), we focus on ambiguity and examine clustering size separately.

6.4 Empirical Evaluation

In this section, we present the experimental setup and results on the clustering hypothesis to answer the research questions defined in the previous section. The code required to reproduce all experiments is available online[2]. All experiments were conducted on a laptop with an Intel® Core™i7-11800H CPU, NVIDIA 3050 Ti Moblie GPU32 GB RAM, and took ∼40h.

6.4.1 Experimental Setup

Case Study Subjects. We performed experiments with 59 regular languages encoded by DFAs. Five of the languages are evaluation subjects from the literature, including three of the Tomita grammars [245] (Tomita 3, 5, and 7), a model of an MQTT server [234], and a regular expression used by Michalenko et al. [152]. Additionally, we randomly generated 30 DFAs and 24 Moore machines with up to 12 states and 72 transitions using AALPY [166]. Furthermore, to perform the analysis for RO3, we encoded context-free languages from [279] as PDAs. Those languages include variations of X^nY^n languages, Dyck languages [97], and variations of Dyck languages. Thus, our experiments cover automata with small state spaces like the Tomita grammars often used in RNN research [271] and with large state spaces spanned by the PDAs, which use stacks of unbounded size.

Training. For each language, we trained an RNN to achieve perfect accuracy for three consecutive epochs on the validation data to potentially increase the likelihood of forming clusters in the hidden state space [151]. We trained Elman RNNs [67] with $tanh$ and $ReLU$ activation functions, LSTMs [95], and

[2] https://github.com/DES-Lab/Clustering_RNN_hidden_state_space

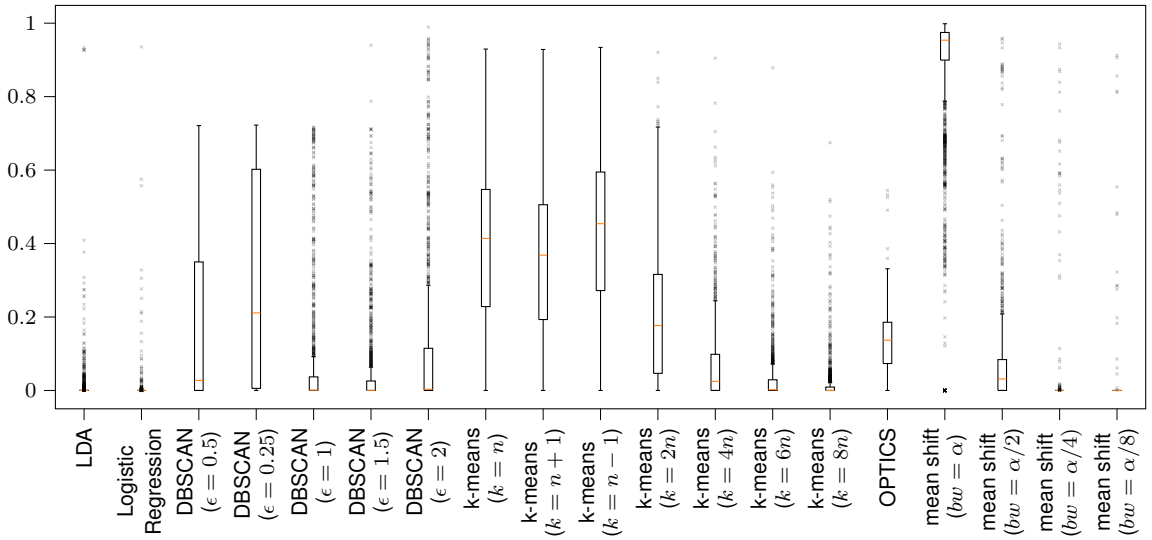


Figure 6.2: Boxplots of the weighted ambiguity resulting from different clustering techniques for all 1350 experiments whose RNNs achieved at least 80% accuracy

GRUs [47], each of them with one layer of size t, one layer of size $1.5t$, two layers of size t, where t is the number of transitions of the ground-truth automaton producing the language under consideration. We have chosen these network sizes since a one-layer Elman RNN with t hidden neurons is sufficient to encode automata with t transitions [11, 86, 153]. Additionally, we also train slightly larger networks, as an increased size may be beneficial for training. We decided to not consider very large networks, as it would complicate the clustering analysis due to dimensionality reduction becoming more important, hindering us from concentrating on clustering approaches. Moreover, the network sizes are sufficient for accurate training. For all experiments, we used the ADAM optimizer [118] with a learning rate of 0.0005. The training data consisted of $n_t = 50k$ randomly sampled words of lengths in the range $[1, 15]$, with labels derived from the ground-truth model. The validation data contained $n_v = 2000$ words, resulting in appr. $10k$ different hidden-state vectors for clustering. For all experiments, we trained two RNNs per configuration (network type, size).

Classification & Clustering. We used LDA and LR to learn classification models to determine whether automaton states can be separated in the hidden state space. Both approaches are supervised classification techniques, to which we provide $\mathcal{HS} \subset H \times Q$, sampled pairs of hidden states and automaton states.

For the clustering algorithms, we apply various parameterizations and use Euclidean distance as a distance metric. We set the k of k-means based on the size $n = |Q|$ of the minimal ground-truth automata A with $k \in \{n-1, n, n+1, 2n, 4n, 6n, 8n\}$. With the first three values, we check whether a good clustering exists that is close to the minimal automaton representation. We also use greater values because an RNN may learn a non-minimal representation. We choose the parameters of DBSCAN and *mean shift* as follows. For DBSCAN we experiment with multiples of the neighborhood size $\epsilon = 0.5$, the default in the scikit-learn library. We leave the other parameter $minNeighbors$ at its default value of 5. For mean shift, we estimate the bandwidth bw with scikit-learn, which we denote α, to perform experiments with multiples of α. As OPTICS improves upon DBSCAN by mitigating its sensitivity on parameter values, we only apply its default parameterization. Since mean shift and OPTICS require more computation time than other techniques, we reduced the sample $\mathcal{H}$ to 25% of its size for these two.

6.4.2 Analysis of RNNs Trained on Regular Languages

In the first set of experiments, we consider all four RNN types, Elman RNNs with ReLU and tanh activation functions, LSTMs, and GRU networks, as well as all clustering techniques. We evaluated all

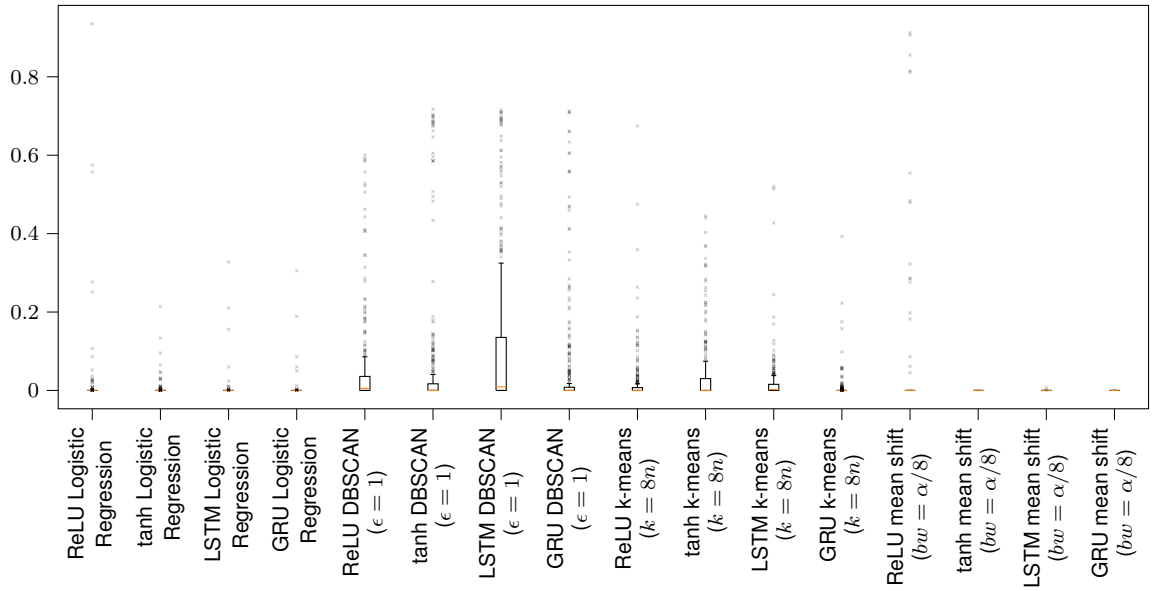


Figure 6.3: Weighted ambiguity for selected methods sorted by network types

Table 6.1: Number of perfect clusterings (zero ambiguity) achieved by selected clustering methods

Clustering Function	LDA	LR	DBSCAN			k-means			OPTICS	mean shift		
Parameters			0.5	0.25	1.5	n	$6n$	$8n$		$\alpha/2$	$\alpha/4$	$\alpha/8$
Regular Languages (1350 experiments)	1003	1235	368	185	639	49	629	783	16	373	1296	1331
PDA stackless (600 experiments)	18	43	7	0	12	0	8	10	0	23	78	260
PDA stackfull (600 experiments)	16	30	7	0	9	4	15	16	0	14	31	81
PDA top-of-stack (600 experiments)	17	35	9	0	7	9	15	16	0	20	32	66

RNNs and only considered those whose accuracy ($agree_{R,L}$ from Section 6.3.2 evaluated on accuracy-validation data) was $\geq 80\%$. We chose this accuracy cutoff as, in practice, RNNs rarely achieve perfect generalization due to the complexity of the underlying task or quality of training data. In total, this resulted in a selection of 1350 from 1416 RNNs.

Figure 6.2 summarizes the results of these experiments. It shows boxplots of the weighted cluster ambiguity $wamb$ resulting from different clustering techniques, with outliers denoted by small crosses. Furthermore, the third row of Table 6.1 shows the number of perfect clusterings, i.e. clusterings with $wamb = 0$, achieved by each method.

We observe that the LDA and LR are able to achieve perfect classification in 74% and 91% of the cases, respectively. LDA has a mean $wamb$ of 0.0009 ± 0.05, while LR's mean $wamb$ was 0.0004 ± 0.039. Relating to **RQ1**, this high level of accuracy indicates that in the majority of considered cases, even with networks that did not achieve perfect accuracy, *piecewise linear separability of the hidden state space is possible*. To further solidify our findings, we perform a quantile test on the $wamb$ with the alternative hypothesis being that 0.95-quantile is less than 0.05, i.e., if the $wamb$ is low in 95% of the cases. Due to the large number of outliers, we failed to reject the null hypothesis, i.e., we could not find statistically significant support for general linear separability. Recall, however, that LDA and LR are supervised multiclass classification techniques derived using additional information about the correspondence between hidden and automaton states, which is usually not available, therefore we examine the ambiguity of unsupervised clustering approaches.

The median weighted ambiguity of DBSCAN with $\epsilon \geq 1$ is almost equal to 0, which means that in at least half of the cases, we can identify semantically meaningful discrete states from clusters. Likewise, k-means achieves a low ambiguity when given at least four times as many clusters as "necessary", i.e.,

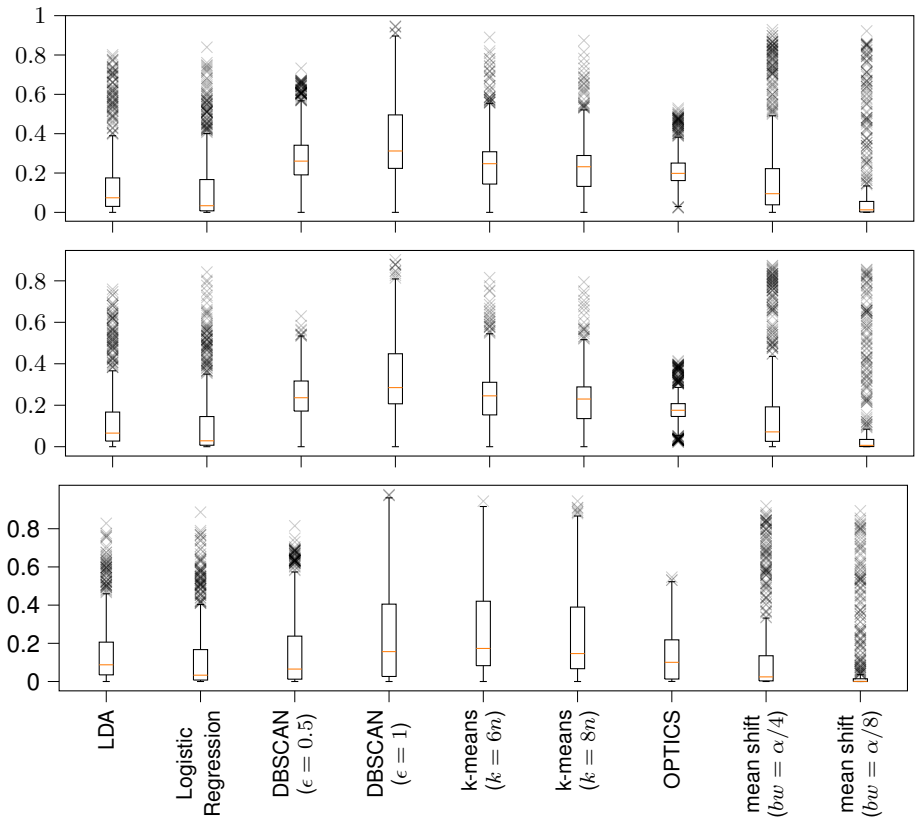


Figure 6.4: Weighted ambiguity for selected methods for RNNs trained on PDAs, without stack information (bottom) and with stack length (middle), and with top of the stack mapping (top)

$k \geq 4n$. This means that RNNs seem to learn non-minimal representations of the concept that is being learned. OPTICS generally improves upon DBSCAN, but in this use case, it seems to perform slightly worse than DBSCAN. The accuracy of clusters computed by mean shift depends on the parameterization of the bandwidth bw. The bandwidth estimated by scikit-learn results in a number of clusters that is smaller than n, causing high ambiguity. But as we decrease bw, in turn increasing the number of found clusters, mean shift becomes the best clustering method, even outperforming supervised classification approaches like LDA and LR. Regarding **RQ2**, we can state that *clusters often correlate with automaton states*, with the caveat that proper parameterization is necessary. As for LDA and LR, we perform a hypothesis test on $wamb$ with the alternative hypothesis that the 0.95-quantile is less than 0.05. The only clustering method for which we rejected the null hypothesis with $p < 0.05$ is mean shift with $bw = \alpha/8$, i.e., we see significant support for the clustering hypothesis to hold in this case.

Impact of Network Architecture. Next, we examine the influence of the RNN architecture on classification and clustering. Figure 6.3 shows the average weighted ambiguity achieved by LR and selected clustering parameterizations for each RNN architecture separately. GRU networks appear to create state space structures that lend themselves best to clustering with any of the approaches. Interestingly, Elman ReLU RNNs lead to the highest ambiguity on average, while Elamn RNNs with $\tanh$ activation function were on average less ambiguous. A potential explanation is that ReLU activations are unbounded, whereas $\tanh$ is bounded. Hence, $\tanh$ Elman RNNs may create clusters in the saturated area of $\tanh$.

6.4.3 Analysis of RNNs Trained on Context-Free Languages

In the second set of experiments, we trained RNNs to recognize context-free languages in the same manner as described in Section 6.4.1. In addition to original RNN sizes, we examined RNNs with $3n$ and $5n$ neurons, where n is the number of transitions found in the ground-truth PDA. The training resulted in 600 networks all of which had an accuracy higher than 80%.

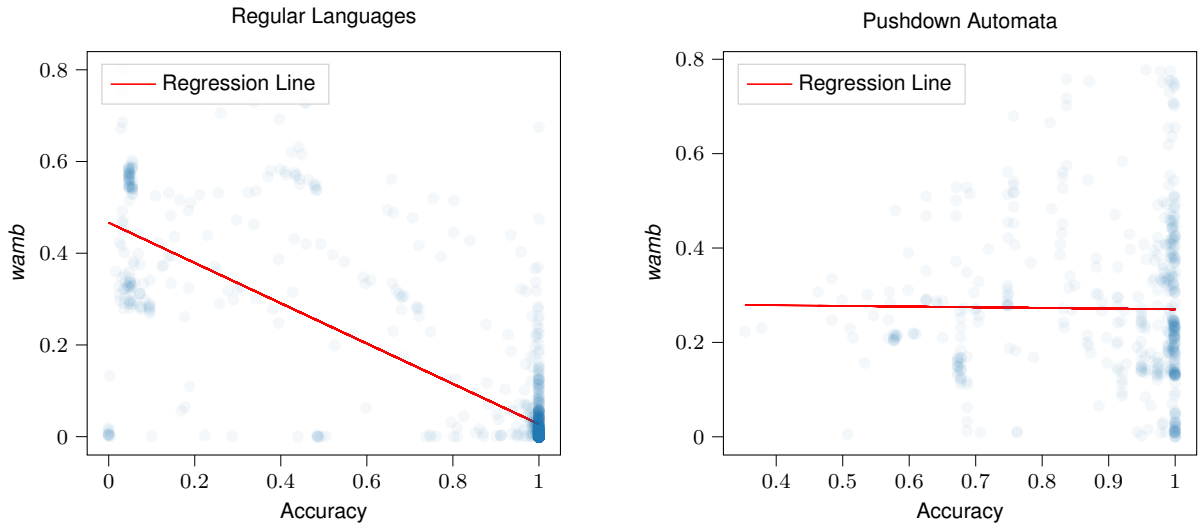


Figure 6.5: Relationship between RNN accuracy and weighted ambiguity for RNNs trained on regular languages (left) and PDAs (right)

Figure 6.4 summarizes the weighted ambiguity results of selected clustering methods. We performed the analysis with three PDA state abstractions presented in Sect. 6.3.2. We considered two stackful mapping, one that considers the PDA location and the symbol found at the top of the stack, and the other that considers the PDA location and the current stack length. In addition, we examined a mapping that ignores the stack and only considers the current PDA location. The stackful abstractions strengthen the analysis, as they offer an alternative mapping that provides more information when compared to cluster-to-location mapping. However, we did not observe major differences between the weighted ambiguity of these abstractions, implying that additional stack information has little influence on RNN states compared to PDA locations themselves.

As seen in Table 6.1, compared to RNNs trained to recognize regular languages, RNNs trained to recognize context-free languages achieved perfect piecewise linear separability in a small fraction of overall experiments. From this, we deduce that hidden state vectors of RNNs trained to recognize context-free languages are rarely perfectly linearly separable and tend to form clusters that carry less semantic meaning than ones computed on RNNs trained on regular languages.

In general, we observe that the clustering functions of RNNs trained to recognize context-free languages are more ambiguous than those of RNNs trained to recognize regular languages, and as seen in Tab. 6.1 rarely achieve perfect clustering. More concretely, while k-means with $8n$ clusters resulted in mappings with low weighted ambiguity (0.018 ± 0.05), it failed to compute unambiguous mappings for all configurations of PDA experiments resulting in weighted ambiguities of 0.245 ± 0.238 for stackless abstraction, 0.234 ± 0.139 for length-of-stack abstraction, and 0.241 ± 0.151 for the top-of-stack abstraction.

To further compare the clustering hypothesis on RNNs trained on regular and context-free languages, we examined the relationship between weighted ambiguity and RNNs accuracy. In Figure 6.5, we plot the weighted ambiguity against the accuracy of every RNN from our experiments. The red lines are linear regression lines. We observe a strong negative correlation between weighted ambiguity and accuracy for RNNs trained on regular languages, while there is no correlation between weighted ambiguity and accuracy for RNNs trained on context-free languages: the Pearson correlation coefficient of -0.802 and -0.012 respectively. This finding further implies that the clustering hypothesis mostly holds for accurate RNNs trained on regular languages, while it cannot be assumed even for accurate RNNs trained on more complex languages.

Regarding **RQ3**, we showed that unlike for regular languages, clusters of hidden state vectors of RNNs trained on context-free languages are generally hardly linearly separable and do not strongly correlate with PDA locations. This can be considered a falsification of the clustering hypothesis on the

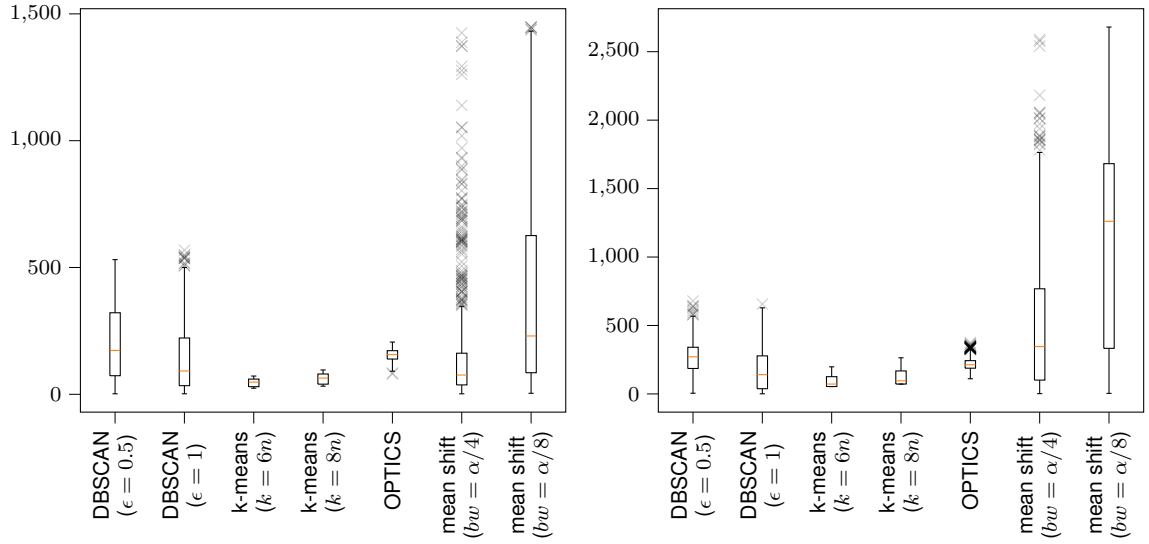


Figure 6.6: Number of clusters found in experiments of RNNs trained on regular languages (left) and context-free languages (right)

higher-level languages, given that it mostly holds for regular languages, but does not hold for the next level in Chomsky hierarchy, that is for context-free languages. However, in rare cases, clustering may be an effective abstraction, as Table 6.1 shows that perfect clustering can be achieved, but only rarely.

6.4.4 Number of Clusters

We only considered ambiguity so far. As clustering is a tool for abstraction, we also need to look at the size of the abstraction, i.e., the number of clusters. K-means fixes this size, but the number of clusters derived by other approaches depends on their parameters *and* the given data. Figure 6.6 shows box plots of the number of clusters derived by different techniques for both previously discussed experiments. Two instantiations of k-means are shown as a reference.

For k-means, the number of clusters is tied to the number of states, as explained in Section 6.4.1. As k-means has a parameterizable number of clusters ($8n$ was sufficient to achieve low ambiguity) and it showed good performance for RNNs trained on regular languages, it is the most viable clustering function for the analysis of RNNs. The number of clusters in DBSCAN increases with decreasing ϵ parameter. While regardless of the ϵ the total number of clusters is higher than in k-means, it still facilitates an abstraction of RNNs. OPTICS finds a similar number of clusters as DBSCAN but with higher weighted ambiguity. The fact that mean shift with $bw = \alpha/8$ finds up to $1.5k$ clusters in $\sim 2.5k$ data points (recall that we reduced the data for mean shift) and that the third quartile in Figure 6.6 (left) is 500 weakens our findings regarding **RQ2**. The highly accurate clusterings found with mean shift hardly group states in many cases, thus making it trivial to achieve high clustering accuracy.

Discussion. Given our findings on the number of clusters, we should consider techniques and parameterizations that create reasonably sized abstractions. K-means with $k = 8n$ achieves perfect clustering in 58% of the cases on RNNs trained on regular languages (see Table 6.1). Additionally, we found that this k-means parameterization achieved a mean $wamb$ of 0.018 when clustering over RNNs trained on regular languages, but in the most extreme case, $wamb$ was 0.67. Hence, clusters mostly correlate with automaton states, if an appropriate clustering technique is applied. As such reasonably-sized clustering functions do not achieve high accuracy in some cases, we cannot rely on the clustering hypothesis alone. This is exacerbated when we move to more complex languages, as we have observed in the experiments with context-free languages. Hence, for safety-critical applications, like autonomous driving, additional techniques are needed to mitigate the error resulting from ambiguous clusters. Probabilistic automata

learning has been shown promising in this regard [64], as it is able to infer additional states that help to mitigate coarse clustering.

Numerical Values of Presented Results

In Table 6.2 we show detailed results from our classification and clustering experiments performed on RNNs trained on regular languages. Detailed values of our experiments over context-free languages are shown in Table. 6.3, Table 6.4, and Table 6.5.

Table 6.2: Clustering ambiguity results from 1350 RNNs that achieved 80% accuracy

Clustering Function	**Ambiguity**		**Weighted Ambiguity**		**Number of Clusters**		**# Perfect Clustering**
	Mean ± Std Dev	Max	Mean ± Std Dev	Max	Mean ± Std Dev	Max	
LDA	0.01 ± 0.06	0.933	0.009 ± 0.05	0.933	8 ± 2.5	15	1003
Logistic Regression	0.003 ± 0.036	0.935	0.004 ± 0.039	0.935	8 ± 2.5	12	1235
DBSCAN ($\epsilon = 0.5$)	0.002 ± 0.002	0.025	0.179 ± 0.239	0.721	200 ± 140	531	368
DBSCAN ($\epsilon = 0.25$)	0.003 ± 0.002	0.02	0.29 ± 0.276	0.772	210 ± 118	506	185
DBSCAN ($\epsilon = 1$)	0.005 ± 0.025	0.672	0.066 ± 0.15	0.716	143 ± 135	568	591
DBSCAN ($\epsilon = 1.5$)	0.016 ± 0.063	0.939	0.05 ± 0.123	0.939	99 ± 115	596	639
DBSCAN ($\epsilon = 2$)	0057 ± 0.144	0.989	0.110 ± 0.208	0.98	69 ± 90	560	514
k-means ($k = n$)	0.359 ± 0.199	0.863	0.395 ± 0.207	0.929	8.5 ± 2.5	12	49
k-means ($k = n + 1$)	0.322 ± 0.131	0.811	0.356 ± 0.201	0.928	9.5 ± 2.5	13	68
k-means ($k = n - 1$)	0.401 ± 0.2	0.895	0.428 ± 0.21	0.934	7.5 ± 2.5	11	42
k-means ($k = 2n$)	0.175 ± 0.154	0.736	0.2 ± 0.172	0.92	16 ± 5	24	209
k-means ($k = 4n$)	0.06 ± 0.091	0.652	0.068 ± 0.105	0.904	33 ± 10	48	432
k-means ($k = 6n$)	0.0291 ± 0.065	0.585	0.032 ± 0.075	0.878	50 ± 15	72	629
k-means ($k = 8n$)	0.017 ± 0.051	0.54	0.0181 ± 0.057	0.674	67 ± 21	96	783
OPTICS	0.006 ± 0.003	0.06	0.134 ± 0.071	0.544	156 ± 20	206	16
mean shift ($bw = \alpha$)	0.847 ± 0.244	0.998	0.878 ± 0.201	0.998	1.35 ± 1.2	20	84
mean shift ($bw = \alpha/2$)	0.038 ± 0.063	0.935	0.073 ± 0.136	0.958	37 ± 51	829	373
mean shift ($bw = \alpha/4$)	0.001 ± 0.014	0.246	0.011 ± 0.086	0.943	136 ± 178	1425	1296
mean shift ($bw = \alpha/8$)	0.0004 ± 0.005	0.09	0.005 ± 0.06	0.912	391 ± 397	1448	1331

Table 6.3: Clustering ambiguity results from 600 RNNs trained to recognize PDAs with top-of-stack mapping

Clustering Function	Ambiguity		Weighted Ambiguity		Number of Clusters		# Perfect Clustering
	Mean ± Std Dev	Max	Mean ± Std Dev	Max	Mean ± Std Dev	Max	
LDA	0.195 ± 0.181	0.777	0.154 ± 0.196	0.800	5 ± 3	11	17
Logistic Regression	0.129 ± 0.152	0.774	0.128 ± 0.188	0.839	5 ± 3	11	35
DBSCAN ($\epsilon = 0.5$)	0.054 ± 0.054	0.249	0.284 ± 0.164	0.732	264 ± 129	667	9
DBSCAN ($\epsilon = 0.25$)	0.028 ± 0.041	0.230	0.293 ± 0.169	0.674	302 ± 109	706	0
DBSCAN ($\epsilon = 1$)	0.104 ± 0.110	0.911	0.359 ± 0.208	0.694	196 ± 140	649	7
DBSCAN ($\epsilon = 1.5$)	0.207 ± 0.226	0.957	0.491 ± 0.250	0.947	95 ± 118	649	14
DBSCAN ($\epsilon = 2$)	0.327 ± 0.283	0.957	0.609 ± 0.249	0.957	50 ± 93	176	8
k-means ($k = n$)	0.355 ± 0.175	0.881	0.403 ± 0.208	0.918	33 ± 26	110	9
k-means ($k = n + 1$)	0.340 ± 0.168	0.857	0.382 ± 0.203	0.913	39 ± 28	120	11
k-means ($k = n - 1$)	0.377 ± 0.185	0.889	0.430 ± 0.217	0.924	27 ± 25	100	6
k-means ($k = 2n$)	0.313 ± 0.161	0.834	0.341 ± 0.193	0.908	66 ± 53	220	11
k-means ($k = 4n$)	0.282 ± 0.150	0.768	0.286 ± 0.173	0.895	133 ± 107	440	15
k-means ($k = 6n$)	0.267 ± 0.146	0.754	0.259 ± 0.160	0.889	200 ± 161	660	15
k-means ($k = 8n$)	0.257 ± 0.141	0.726	0.241 ± 0.151	0.873	267 ± 215	880	16
OPTICS	0.067 ± 0.04	0.198	0.215 ± 0.106	0.529	224 ± 55	373	0
mean shift ($bw = \alpha$)	0.637 ± 0.266	0.938	0.732 ± 0.215	0.938	2 ± 4	40	1
mean shift ($bw = \alpha/2$)	0.279 ± 0.177	0.932	0.435 ± 0.259	0.932	42 ± 76	813	20
mean shift ($bw = \alpha/4$)	0.078 ± 0.070	0.643	0.199 ± 0.246	0.928	512 ± 504	2577	32
mean shift ($bw = \alpha/8$)	0.024 ± 0.038	0.34	0.102 ± 0.211	0.922	1130 ± 734	2671	66

Table 6.4: Clustering ambiguity results from 600 RNNs trained to recognize PDAs with location $\times$ length of stack mapping

Clustering Function	Ambiguity		Weighted Ambiguity		Number of Clusters		# Perfect Clustering
	Mean ± Std Dev	Max	Mean ± Std Dev	Max	Mean ± Std Dev	Max	
LDA	0.169 ± 0.127	0.685	0.141 ± 0.176	0.759	5 ± 3	11	16
Logistic Regression	0.105 ± 0.119	0.847	0.121 ± 0.181	0.842	5 ± 3	11	30
DBSCAN ($\epsilon = 0.5$)	0.036 ± 0.042	0.192	0.25 ± 0.136	0.63	264 ± 129	679	7
DBSCAN ($\epsilon = 0.25$)	0.017 ± 0.031	0.178	0.264 ± 0.14	0.624	302 ± 109	684	0
DBSCAN ($\epsilon = 1$)	0.078 ± 0.093	0.879	0.342 ± 0.185	0.901	196 ± 140	656	4
DBSCAN ($\epsilon = 1.5$)	0.184 ± 0.226	0.965	0.456 ± 0.237	0.965	95 ± 119	641	9
DBSCAN ($\epsilon = 2$)	0.3700 ± 0.281	0.965	0.578 ± 0.237	0.965	51 ± 49	709	5
k-means ($k = n$)	0.341 ± 0.147	0.742	0.404 ± 0.184	0.866	16 ± 8	33	4
k-means ($k = n + 1$)	0.324 ± 0.144	0.735	0.383 ± 0.183	0.853	19 ± 8	36	5
k-means ($k = n - 1$)	0.371 ± 0.148	0.749	0.441 ± 0.179	0.875	13 ± 8	30	2
k-means ($k = 2n$)	0.291 ± 0.139	0.794	0.336 ± 0.177	0.158	33 ± 17	66	2
k-means ($k = 4n$)	0.256 ± 0.128	0.684	0.279 ± 0.16	0.822	66 ± 33	132	11
k-means ($k = 6n$)	0.242 ± 0.124	0.665	0.252 ± 0.145	0.815	98 ± 50	198	15
k-means ($k = 8n$)	0.233 ± 0.121	0.627	0.234 ± 0.139	0.794	131 ± 66	264	16
OPTICS	0.048 ± 0.035	0.179	0.183 ± 0.087	0.414	224 ± 55	376	0
mean shift ($bw = \alpha$)	0.611 ± 0.252	0.914	0.707 ± 0.201	0.914	3 ± 4	46	0
mean shift ($bw = \alpha/2$)	0.242 ± 0.147	0.839	0.395 ± 0.246	0.876	44 ± 81	783	14
mean shift ($bw = \alpha/4$)	0.066 ± 0.065	0.576	0.179 ± 0.238	0.873	513 ± 512	2589	31
mean shift ($bw = \alpha/8$)	0.02 ± 0.04	0.381	0.090 ± 0.2	0.854	1123 ± 738	2680	81

6.4.5 Additional Findings

In this section, we present some additional findings related to the clustering hypothesis. We choose to display these results after the analysis of the research questions, as it extends our analysis presented in previous sections but does not change our findings with respect to the research questions.

Table 6.5: Clustering ambiguity results from 600 RNNs trained to recognize PDAs with stackful mapping

Clustering Function	Ambiguity		Weighted Ambiguity		Number of Clusters		# Perfect Clustering
	Mean ± Std Dev	Max	Mean ± Std Dev	Max	Mean ± Std Dev	Max	
LDA	0.177 ± 0.163	0.849	0.160 ± 0.18	0.808	5 ± 3	11	18
Logistic Regression	0.131 ± 0.179	0.891	0.133 ± 0.196	0.886	5 ± 3	11	43
DBSCAN ($\epsilon = 0.5$)	0.014 ± 0.023	0.186	0.171 ± 0.217	0.815	264 ± 130	679	7
DBSCAN ($\epsilon = 0.25$)	0.007 ± 0.018	0.176	0.185 ± 0.229	0.697	302 ± 109	705	0
DBSCAN ($\epsilon = 1$)	0.046 ± 0.098	0.957	0.246 ± 0.235	0.978	169 ± 140	650	7
DBSCAN ($\epsilon = 1.5$)	0.137 ± 0.226	0.988	0.376 ± 0.286	0.988	95 ± 118	637	12
DBSCAN ($\epsilon = 2$)	0.238 ± 0.288	0.989	0.5 ± 0.29	0.99	50 ± 93	710	11
k-means ($k = n$)	0.396 ± 0.238	0.979	0.476 ± 0.25	0.979	5 ± 3	11	0
k-means ($k = n + 1$)	0.364 ± 0.225	0.966	0.447 ± 0.244	0.961	6 ± 3	12	0
k-means ($k = n - 1$)	0.469 ± 0.265	0.986	0.534 ± 0.262	0.986	4 ± 3	10	0
k-means ($k = 2n$)	0.302 ± 0.221	0.96	0.386 ± 0.249	0.953	11 ± 6	22	4
k-means ($k = 4n$)	0.235 ± 0.215	0.945	0.309 ± 0.25	0.947	22 ± 11	44	5
k-means ($k = 6n$)	0.201 ± 0.205	0.937	0.271 ± 0.245	0.944	33 ± 17	66	8
k-means ($k = 8n$)	0.181 ± 0.198	0.925	0.245 ± 0.238	0.944	44 ± 22	88	10
OPTICS	0.019 ± 0.03	0.189	0.134 ± 0.147	0.546	224 ± 55	383	0
mean shift ($bw = \alpha$)	0.567 ± 0.284	0.952	0.649 ± 0.251	0.946	3 ± 4	36	2
mean shift ($bw = \alpha/2$)	0.817 ± 0.181	0.931	0.345 ± 0.266	0.931	42 ± 75	782	23
mean shift ($bw = \alpha/4$)	0.027 ± 0.052	0.525	0.141 ± 0.234	0.919	509 ± 505	2619	78
mean shift ($bw = \alpha/8$)	0.008 ± 0.034	0.665	0.079 ± 0.192	0.893	1120 ± 735	2699	260

Clustering of GRU Networks

In the following, we examine the clustering hypothesis for GRU networks trained on regular languages, for which we found particularly low ambiguity measurements. Figure 6.7 shows box plots of ambiguity values measured for different clustering techniques. On the left, we show measurements from all experiments involving GRUs and on the right, we show measurements from experiments, where LDA is able to perfectly separate hidden states corresponding to all automaton states.

We found that the hidden state space of the trained GRUs has a structure that is well-suited for linear separation of states. In 88% of the experiments, both LDA and LR can perfectly separate states.

This benefits k-means as well, which achieves lower ambiguity in all considered parameterizations when compared to other network types. In particular, k-means with $k = 8n$ performs very well, where the third quartile of the ambiguity values is equal to zero. From the other techniques, DBSCAN with $\epsilon = 1$ benefits from restricting the analysis to GRUs. OPTICS and especially mean shift are hardly affected. Focusing only on the experiments where LDA has an ambiguity of 0 (Figure 6.7 (right)), we see that k-means performs especially well, with an ambiguity of at most 0.034 for $k = 8n$.

Training of Noisy Correct-by-Construction RNNs

The training of RNNs is governed by a large number of parameters and aspects, which potentially affect clustering. RNN training parameters and initialization potentially affect clustering, therefore we investigate a different training initialization. We create RNNs from ground-truth automata as described below. These RNNs are guaranteed to form dense clusters. After introducing noise into the RNN weights, we will examine if training adjusts the weights in a way that again leads to well-formed, meaningful clusters.

RNN Construction. Our construction of RNNs to encode automata is similar to those construction methods proposed in [11, 86, 153]. We use Elman RNNs with $tanh$ activation and a single layer, which we use in the saturated area of $tanh$, where it works like a threshold gate. Such RNNs are created so that they conform exactly to the automaton. Their hidden-state space is saturated and forced into clusters that correspond to the states and transitions of the automaton. We will examine the following: after

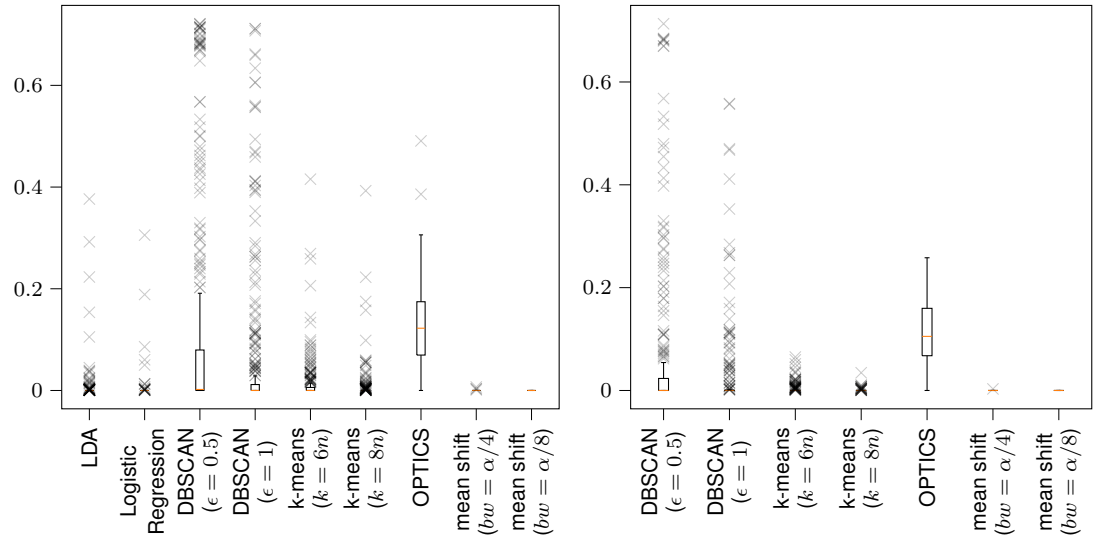


Figure 6.7: Boxplots of the weighted ambiguity resulting from different clustering techniques for all 346 experiments with GRUs (left) and for a subset of 278 GRU experiments, where LDA and LR achieved perfect separability (right)

introducing Gaussian noise in the correct-by-construction RNNs weights and retraining them, does their hidden-state space still gravitate toward the initially specified, "perfect" conditions? That is, we will examine if training causes clusters to form again after the introduction of the noise changes the hidden state space.

This set of experiments was performed on all DFAs, with a single RNN size and type, and three configurations per regular language with varying saturation and noise parameters. We considered only RNNs that achieved 100% accuracy after retraining, thus reducing this set of experiments to 95 RNNs.

The results can be seen in Figure 6.8. Both LDA and LR were able to perfectly classify hidden-state vectors in 92% and 95% percent of the cases, respectively. DBSCAN managed to compute clusters with low ambiguity, but compared to the previous set of experiments, it achieved so with smaller ϵ. On average, OPTICS found clusters with very low ambiguity (0.003 ± 0.0008), but interestingly it found perfect clusterings in only 17% of the cases. K-means had similar performance as in the previous set of experiments, with $k = 6n$ and $k = 8n$ achieving near-perfect results. Once again, mean shift outperformed both LDA and LR, as well as other unsupervised clustering approaches. We therefore conclude that *initial conditions do affect clustering*. For example, our constructed weights seem to lead to smaller distances in the state space, such that DBSCAN performed better with smaller ϵ.

Relationship between Accuracy and Ambiguity during the Training Process

We previously analyzed the relationship between accuracy and ambiguity with already trained networks, as seen in Figure 6.9, while in this section we consider the ambiguity over the training process itself. We display the results of this analysis in Figure 6.9. We observe that the ambiguity decreases during the training process, as the accuracy increases. This shows that the clustering hypothesis is related to RNN accuracy, and even further validates the ambiguity metric introduced in Section 6.3.2. In only a few cases, accuracy decreases slightly as ambiguity decreases and vice versa, which explains the kinks in the graphs. However, there is generally a strong negative correlation, i.e., the more accurate an RNN gets, the clustering gets less ambiguous: the Spearman correlation coefficients are -0.87, -0.96, and -0.9 for Elman RNNs, LSTMs, and GRUs in Figure 6.9. Additionally, if the RNN retains the accuracy over long sequences (100-200 input symbols), the clustering ambiguity remains constant.

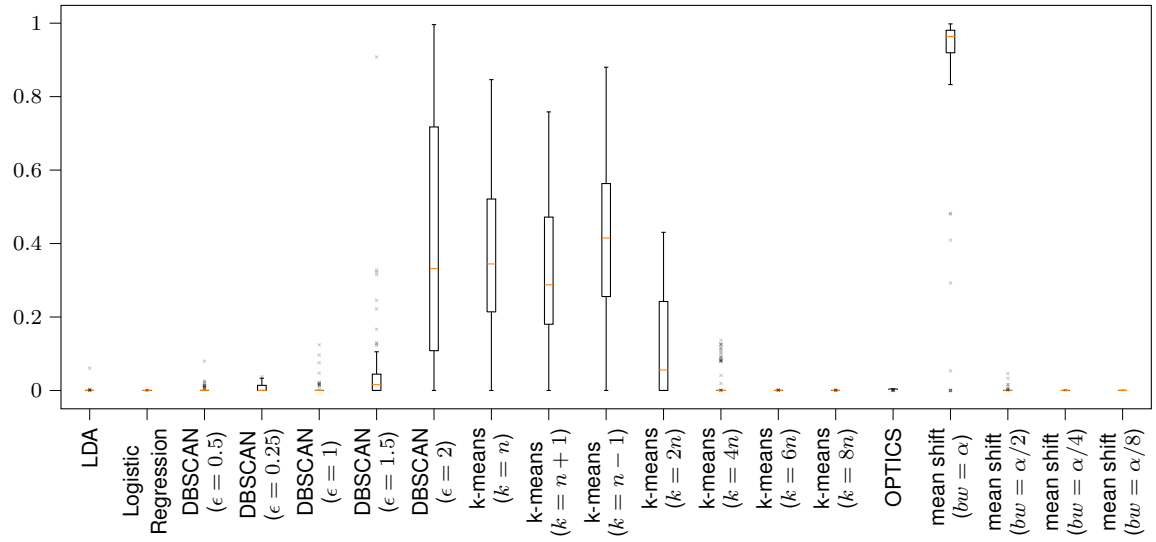


Figure 6.8: Boxplots of the clustering ambiguity resulting from different clustering techniques from 95 trained noisy constructed RNNs

Extraction of Cluster Automata.

Figure 6.10 depicts a non-minimal automaton extracted from an RNN trained on the Tomita 5 grammar. The extracted model is a non-minimal representation of the ground truth model. We have applied the extraction method found in [98] with multiple clustering functions that achieved perfect accuracy, where Figure 6.10 is based on k-means clustering with $k = 4n$. Basically, we create one DFA state for each cluster $k \in K$. For the transitions, suppose that (h, q) and (h', q') are pairs of hidden states and automaton states collected consecutively while processing symbol e. In this case, we add a transition from $c(h)$ to $c(h')$ labeled by e, where c is the clustering function. If the RNN outputs $true$ for one hidden state in a cluster k, we add k to the accepting states.

While the extracted model size varied, all extracted automata were non-minimal representations of the ground-truth model that was used to train the RNN. Our findings conform to those of [98] and they indicate that RNNs learn non-minimal representations of regular languages.

6.4.6 Threats to Validity

To the best of our knowledge, our empirical analysis has two potential limitations: a) statistical significance of results, and b) dependence of our findings on the quality of clustering metric introduced in Section 6.3.2.

Regarding a), we applied exploratory data analysis to find trends in the data from a total of 1950 trained RNNs over 74 (59 regular languages and 15 context-free) formal languages. These networks include different architectures of various sizes, and for each trained network, we conduct experiments that contain multiple parameterizations of clustering algorithms. We examined selected findings through statistical tests, where we found statistically significant support for mean shift computing clusters with low ambiguity and a statistically significant correlation between ambiguity and accuracy of RNNs trained on regular languages. In cases, where trends were very clear, we did not perform statistical tests like for the high ambiguity results involving context-free languages.

Regarding b), we believe that our entropy-based metric for the clustering quality reflects the quality of clustering in the scope of the presented research questions, that is, it encodes the correspondence of identified clusters to states. We further compared it to NMI, and the reason why NMI is not suited as a measure of clustering quality in our research context, is that it might penalize perfect, but non-minimal, correspondence of clusters to states. In addition, since we consider formal languages in our experiments,

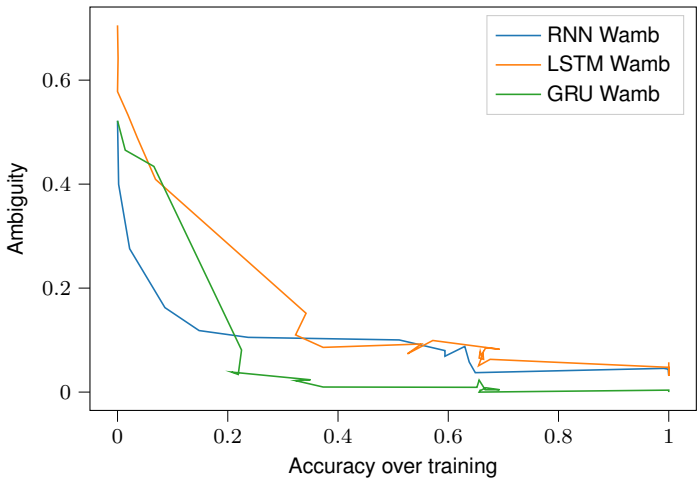


Figure 6.9: Relationship between accuracy and ambiguity over the training process

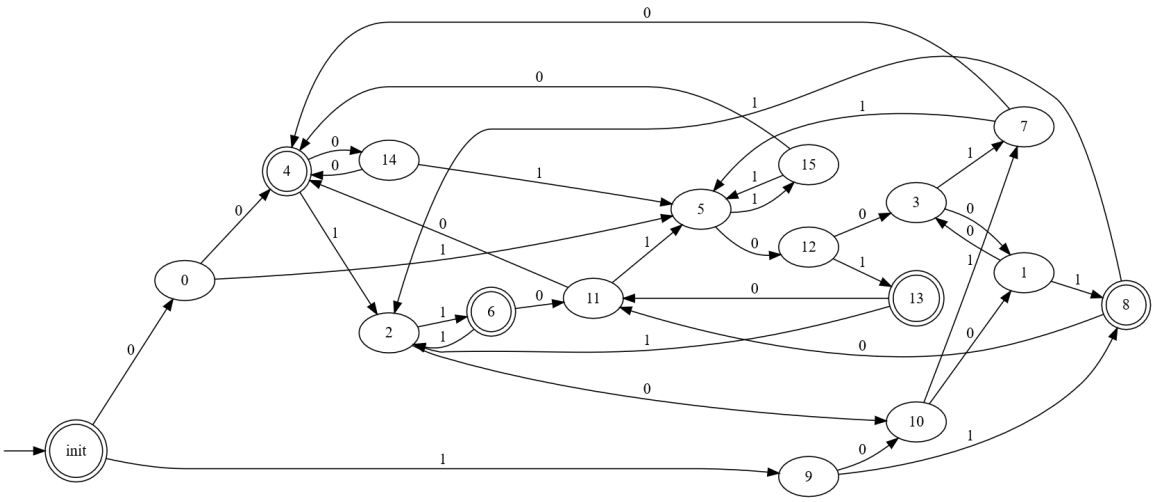


Figure 6.10: Extracted cluster automata from an RNN trained on the Tomita 5 grammar. K-means clustering with $k = 4n$ was used. The extracted model is a non-minimal representation of the ground truth model shown.

our findings might not translate to NLP tasks. However, we believe that moving from context-free to free-form natural language, we may see an even larger ambiguity gap than between regular and context-free languages. However, since a formal model of a natural language does not exist, we cannot examine the clustering hypothesis in that setting, and can only postulate based on this inductive step.

6.5 Concluding Remarks

In this chapter, we empirically analyzed a long-assumed hypothesis, which asserts that RNN hidden-state vectors form clusters that correspond to automaton states. In our analysis, we examined the clustering hypothesis for RNNs that were trained to recognize formal languages. This provided us with a ground truth that allowed us to compare the identified clusters with the original finite-state semantics that underlie the learned concept. Additionally, we examined the (linear) separability of hidden-state vectors into regions corresponding to automata states.

RQ 2.2 What is an appropriate abstraction for the modeling of RNNs?

In our experiments, we observed that the hidden-state vectors of RNNs trained to recognize regular languages can be (piecewise linearly) separated. This finding indicates that it is potentially possible to derive meaningful mappings from hidden-state vectors to automata states. Considering unsupervised clustering, the regions identified by classifiers may contain multiple clusters and clusters may span across decision boundaries. Restricted to regular languages, we show that it is possible to compute clustering functions that correlate with automata states. For example, clusterings obtained by k-means with large k overall achieved high accuracy and perfect accuracy in 58% of the first set of experiments. This drastically changes when considering context-free languages, where the k-means with a large k achieved perfect clustering in only 2.6% of the cases. Since the clustering hypothesis can be assumed for RNNs trained on regular languages, but not for RNNs trained on context-free languages, we postulate that the clustering hypothesis does not hold for any higher-level languages or natural languages. This is a strong indication that the clustering hypothesis alone should not be used as a basis for abstraction in the analysis of RNNs.

7

Extending Reinforcement Learning with Automata Learning

Declaration of Sources

The content presented in this chapter is based on our paper "Reinforcement Learning Under Partial Observability Guided by Learned Environment Models" which was presented at iFM2023 [169].

7.1 Motivation

In this chapter, we show how automata learning can aid reinforcement learning (RL) in the presence of partial observability. Partial observability presents a challenge to RL, which naturally arises in various control problems. Unreliable or inaccurate sensor readings may provide incomplete state information, e.g., static images provided by visual sensors do not capture the agent's movement trajectory and speed. Formally, partial observability occurs when observations of the environment do not allow us to determine the environment's state directly. In such settings, optimal control based on observations only is generally impossible.

For this reason, RL methods often include some form of memory to cope with partial observability, such as the hidden state in recurrent neural networks [92] or fixed-size memory obtained by concatenating previous observations [155]. In this chapter, we propose a method for RL in partially observable environments that can be modeled with partially observable MDPs (POMDPS). The proposed method combines Q-learning [269] with IoAlergia [144]. With IoAlergia, we regularly learn and update MDPs based on the experiences of the RL agent. The learned MDPs approximate the dynamics of the partially observable Markov decision process (POMDP) underlying the environment and their states extend the observation space of Q-learning. To enable this extension, we trace every step of the RL agent on the most recently learned MDP and add the explored MDP state as an observation. Hence, we provide memory by tracking learned environmental states. With this approach, we follow the tradition of state estimation in RL under partial observability [50, 148]. In comparison to earlier work, we overcome strict assumptions on the underlying POMDP, such as knowledge about the number of states, e.g., criticized by Singh et al. [225].

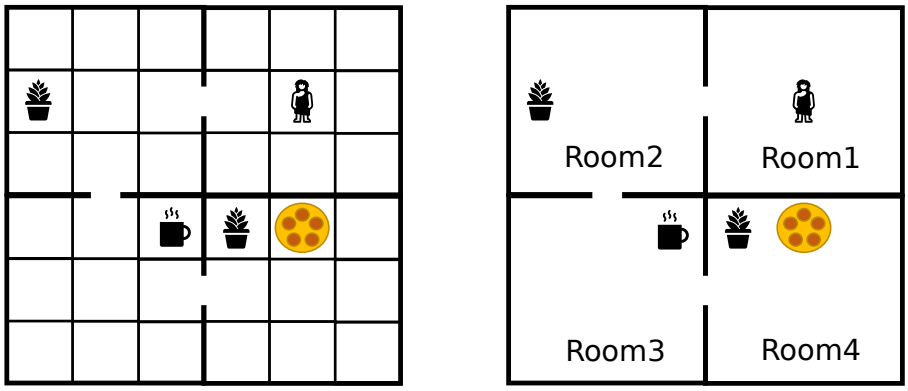


Figure 7.1: Fully and partially observable slippery *OfficeWorld*. With full observations (left), the agent can observe *(x,y)* coordinates, while with the partial observability (right) it can observe only the room number. The agent may slip in some locations, changing the target position of a move.

7.1.1 Guiding Idea

We demonstrate the challenge of RL in partially observable environments on the example shown in Figure 7.1. On the left side of Figure 7.1 we show an example of a fully observable environment and a partially observable counterpart on the right. In both settings, the agent's aim is to navigate through the rooms and reach the cookie. In the fully observable variant, the agent is aware of its position in the form of a tuple (x, y) of coordinates, with the top left corner being the origin $(0, 0)$. From each position, the agent can move in all four directions (top, right, down, left), and such a chosen action stochastically succeeds as expected or not. So the intuitive action sequence to reach the cookie in $(4, 3)$ from the initial position $(4, 1)$ would be the sequence $\langle$left, left, left, down, down, down, right, right, right, up$\rangle$. Since the agent perceives its new position this case can be seen as a fully observable MDP with a known structure but unknown (stochastic) transition probabilities. The agent can also always re-evaluate and re-plan, due to being aware of the exact position after each action.

In the partially observable scenario on the right, the agent only knows in which room it currently is (but not in which of the nine positions compared to the other variant). Consequently, the agent can neither infer its exact position nor the appropriate action just from this observation. It could count, for example, how often it moved in a certain direction in order to enable an educated guess (it would still be a guess due to the stochastic success of actions). In order to enable RL in such an environment, we propose to learn an abstract stochastic model of the environment, such as to capture unobservable state information. The observation space available to the RL agent is then extended with the learned model, which in turn enables RL in partially observable environments.

Structure. This chapter is structured as follows: We start by introducing partially observable formalisms in Section 7.2. In Section 7.3 we present an approach for RL under partial observability aided by automata learning, which we term Q^A-learning. We evaluate the proposed approach against three baseline deep RL methods with fixed memory and three RL methods with LSTMs providing memory in Section 7.4, and conclude the chapter in Section 7.5.

7.2 Background on Partial Observability

Throughout this chapter we rely on standard definitions of MDPs and RL outlined in Chapter 2. In the following, we present MDP adaptations that enable the modeling of partially observable environments.

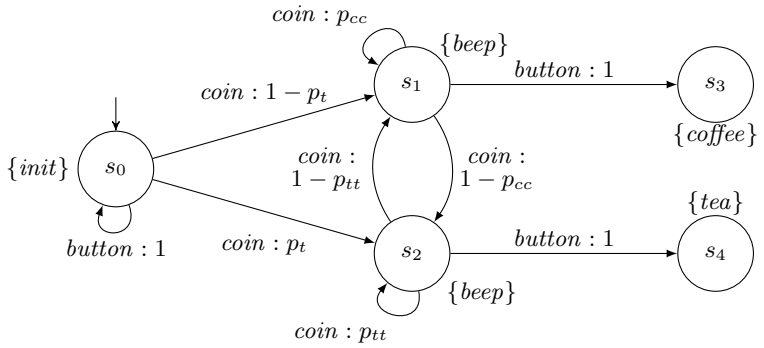


Figure 7.2: A POMDP producing hot beverages

7.2.1 Partially Observable Markov Decision Processes

Partially observable Markov Decision Processes (POMDPs) are a generalization of deterministically-labeled MDPs defined in Section 2.1.2. Compared to deterministically-labeled MDPs, POMDPs are not deterministically labeled, that is, they do not necessarily assign at most one successor to each state and input-output pair.

Definition 7.1 (Partially observable Markov decision processes).
A POMDP is a labeled MDP $\mathcal{M}$, where $\mathcal{M} = \langle Q, I, O, \delta, \lambda, q_0 \rangle$ that does not satisfy the determinism property, as defined in Section 2.1.2.

As with deterministically labeled MDPs, we assume POMDPs to support all actions in all states, s.t. δ is total. Alternative POMDP definitions including probabilistic observation functions can also be accounted for [45].

Example 7.1 (Hot Beverage POMDP). Figure 7.2 shows a POMDP of a vending machine that, depending on parameterized probabilities, produces either tea or coffee. This model is partially observable, as from the initial state $s0$ the input *coin* can lead to two distinct states s_1 and s_2, both of which are labeled with *beep*. Therefore, purely on the input-output observations alone, we cannot conclude which of these two states is reached after the execution of *coin*.

For the individual states s_i, we show the respective observation in curly braces. For each probabilistic transition reported (for the ease of presentation we omit the transitions from s_3 and s_4, but they would loop back to s_0 for any action) we show a corresponding edge, labeled by the action and the transition's probability. While the parameterized probabilities will become more important later on, let us for now assume $p_t = 0.5$, $p_{cc} = 0.9$, and $p_{tt} = 0.1$. In the initial state s_0, for the action $coin$, we would now progress to *either* s_1 *or* s_2, where the resulting observation would be $beep$ for both. Pressing a $button$, we would then move to s_3 or s_4 receiving either a $coffee$ or tea. Alternatively, we can add another $coin$ to move to s_1 with a probability of 0.9 for increasing the chances of getting a $coffee$.

Paths, Traces & Policies. The interaction of an agent with its environment can be described by a *path* that is defined by an alternating sequence of states and actions $s_0 \cdot a_1 \cdot s_1, \cdots, s_n$ starting in the initial state. We denote the set of paths in an MDP $\mathcal{M}$ by $Paths_{\mathcal{M}}$. For partially observable scenarios, *traces* basically replace the states in a path with the corresponding observations. We correspondingly lift observation functions λ to paths, applying λ on every state to derive trace $\lambda(p) = \lambda(s_0) \cdot a_1 \cdot \lambda(s_1)$ from path $p = s_0 \cdot a_1 \cdot s_1$. An agent selects actions based on a *policy* that is a mapping from $Paths_{\mathcal{M}}$ to distributions over actions $Dist(A)$. If a policy σ depends only on the current state, we say that σ is memoryless. With policies relating to action choices, an MDP controlled by a policy defines a probability distribution over paths. Note that in RL, we usually consider memoryless policies—enabled by the assumption of a Markovian environment (modeled as MDP) which guarantees that there is an optimal, memoryless policy for maximizing the reward. With partial observability it is impossible to precisely identify the

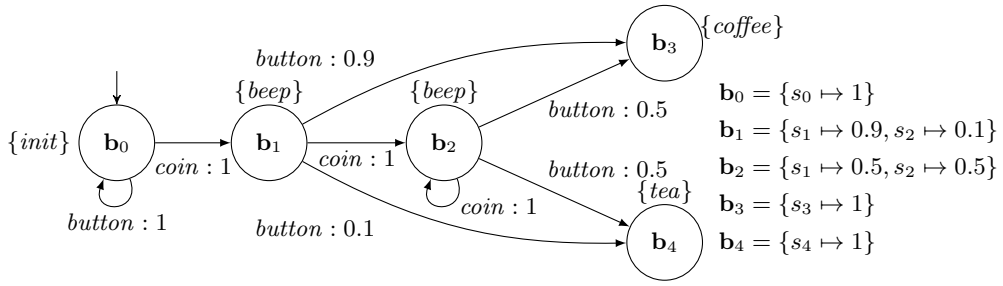


Figure 7.3: A finite BMDP for the POMDP from Figure 7.2 and parameters $p_t = 0.1$ and $p_{tt} = p_{cc} = 0.5$. For brevity, we do not show transitions from $\mathbf{b}_3$ and $\mathbf{b}_4$

.

current state (consider the Hot Beverage example and $s1$ vs. $s2$), meaning that creating optimal policies for POMDPs entails taking the history of previous actions into account—rendering the problem non-Markovian. Alternatively, deriving policies under partial observability can be approached by creating belief-MDPs from POMDPs [43].

7.2.2 Belief-MDPs

As defined in Section 2.1.2, deterministic labeled MDPs are MDPs that adhere to a determinism property, which guarantees that for every observation trace, there is exactly one state that can be reached. Belief-MDPs (BMDP) are deterministic labeled MDPs that represent the dynamics of POMDPs on the basis of observations. These MDPs are defined over so-called belief states, which are probability distributions over states of the POMDP. Belief states essentially denote the states that the corresponding POMDPs could be in after a particular trace. Given an observation trace, we can determine a unique state in a belief MDP, giving us a distribution over possible POMDP states.

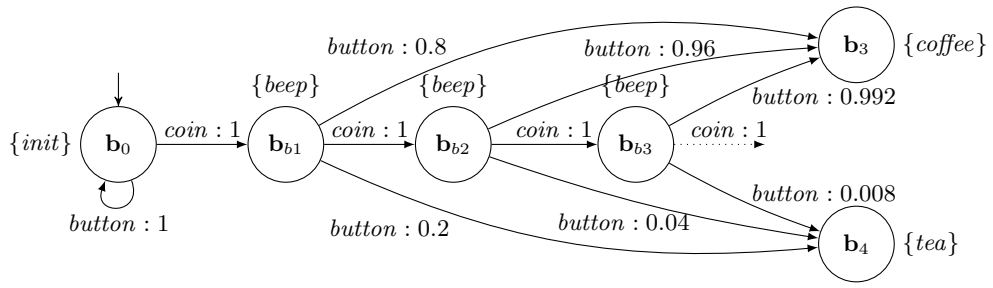
For introducing Belief MDPs (BMDPs), let us consider some auxiliary definitions: Let $P = \langle Q, I, O, \delta, \lambda, q_0 \rangle$ be a POMDP. This defines the beliefs as the set $B = \{\mathbf{b} \in \mathit{Dist}(Q) | \forall s, s' \in \mathit{supp}(\mathbf{b}) : \lambda(s) = \lambda(s')\}$, where *supp*() returns the support of a probability distribution. The probability of observing $o \in O$ after executing $a \in I$ in $s \in Q$ is defined as $\mathbf{P}(s, a, o) = \sum_{s' \in Q, \lambda(s')=o} \delta(s, a, s')$ and in a belief state $\mathbf{b}$ it is $\mathbf{P}(\mathbf{b}, a, o) = \sum_{s \in Q} \mathbf{b}(s) \cdot \mathbf{P}(s, a, o)$. If $\lambda(s') = o$, the subsequent belief update is defined as $[\![\mathbf{b}|a, o]\!](s') = \frac{\sum_{s \in Q} \mathbf{b}(s) \cdot \delta(s,a,s')}{\mathbf{P}(\mathbf{b},a,o)}$.

Definition 7.2 (Belief MDPs).
The BMDP for a POMDP P is a deterministically-labeled MDP $\mathcal{M}_B = \langle B, I, O, \delta_B, \lambda_B, q_0 \rangle$, with B as defined above, $s_{0B} = \{s_0 \mapsto 1\}$, $\lambda_B(\mathbf{b}) = \lambda(s)$ for an $s \in \mathit{supp}(\mathbf{b})$, and

$$\delta_B(\mathbf{b}, a, \mathbf{b}') = \begin{cases} \mathbf{P}(\mathbf{b}, a, \lambda(\mathbf{b}')) & \text{if } \mathbf{b}' = [\![\mathbf{b}|a, \lambda(\mathbf{b}')]\!], \\ 0 & \text{otherwise.} \end{cases}$$

BMDPs allow to synthesize policies under partial observability, i.e., it was shown that an optimal policy for a BMDP is optimal also for the corresponding POMDP [43]. Since they are Markovian, there are further methods for synthesizing memoryless policies. Please note that while in principle there are finite BMDPs (e.g., Figure 7.3), in general they are of infinite size [32] (see, e.g., Figure 7.4).

Example 7.2 (Hot Beverage BMDPs). Let us consider again the POMDP from Figure 7.2 and parameters $p_t = 0.1$ and $p_{tt} = p_{cc} = 0.5$. The corresponding finite BMDP is shown in Figure 7.3. Now suppose that we get a reward for observing *tea*, i.e., when reaching b_4. The BMDP then supports a memoryless policy where we choose the actions *coin* in $\mathbf{b}_0$ and $\mathbf{b}_1$, and *button* in $\mathbf{b}_2$. That is, unless the discounting factor γ is very small (like 0), choosing *button* in $\mathbf{b}_1$ would be optimal for maximizing the immediate reward. A infinite BMDP for parameters $p_t = 0.2$, $p_{tt} = 0.2$ and $p_{cc} = 1$ is shown in Figure 7.4.



$\mathbf{b}_0 = \{s_0 \mapsto 1\}$ $\mathbf{b}_{b1} = \{s_1 \mapsto 0.8, s_2 \mapsto 0.2\}$

$\mathbf{b}_{b2} = \{s_1 \mapsto 0.96, s_2 \mapsto 0.04\}$ $\mathbf{b}_{b3} = \{s_1 \mapsto 0.992, s_2 \mapsto 0.008\}$

$\mathbf{b}_3 = \{s_3 \mapsto 1\}$ $\mathbf{b}_4 = \{s_4 \mapsto 1\}$

Figure 7.4: An infinite BMDP for the POMDP from Figure 7.2 for parameters $p_t = 0.2$, $p_{tt} = 0.2$, and $p_{cc} = 1$. For ease of presentation, we do not show transitions from $\mathbf{b}_3$ and $\mathbf{b}_4$

.

7.2.3 Influence of Partial Observability on Stochastic Automata Learning

In Section 2.3.2 and Chapter 4 we have shown how a model that captures an input-output behavior of a stochastic SUL can be learned with L^*_{SMM} and IOALERGIA. However, so far we have assumed that an MDP underlying SUL's behavior is deterministically labeled. In the remainder of this section, we briefly explain how partial observability affects active and passive learning of a stochastic SUL.

Active Learning of POMDPs. Active learning of POMDPs with L^*_{SMM} usually converges to the finite deterministic approximation of the underlying MDP as shown in Figure 7.3. However, in the perfect learning setting which was defined by Tappler et al. [237], the L^*_{SMM} would not halt as learning would result in the deterministic approximation of ever-increasing size, akin to one shown in Figure 7.4. In this setting, the learning algorithm receives perfect information about all available outputs and corresponding probabilities after the execution of an input-output trace. However, this assumption is generally not assumed for sampling-based algorithms like L^*_{SMM} and L^*_{MDP}, but was rather used by Tappler et al. during the development and construction of the formal proof of L^*_{MDP}.

Passive Learning of POMDPs. In the passive learning setting the learning algorithm always returns a deterministic approximation of the POMDP used to generate the dataset. With IOALERGIA, the infinite learning loop is not possible, since the amount of data provided to the learning algorithm is finite. Learning of the POMDP producing hot beverages shown in Figure 7.2 with IOALERGIA results in the MDP shown in Figure 7.3. If the amount of available traces is increased, the learning algorithm might return a more precise model, akin to one shown in Figure 7.4 (modulo the infinite continuation). However, to get an even more accurate model, the dataset would have to be greatly extended, as we observe that with each addition of a coin, the *tea* probability rapidly converges to 0. Consequently, the learning algorithm would struggle to continue differentiating states even with a substantially larger dataset due to such small transition probabilities.

7.3 Reinforcement Learning Assisted by Automata Learning

In this section, we present Q^A-learning, an approach for reinforcement learning under partial observability. First, we describe the setting and the general intuition behind the approach. Then, we present the state space perceived by the learning agent, followed by a presentation of the complete approach.

Note that we use the terms actions and inputs interchangeably since the term action is commonly used in RL to represent inputs in an underlying MDP. In the same fashion, we use the terms observations and outputs interchangeably.

7.3.1 Overview

Setting. We consider reinforcement learning in partially observable environments. That is, we assume that the environment behaves like a POMDP, where we cannot observe the state directly. Moreover, we do not assume to have a POMDP model of the environment. Initially we only know the available actions and as we learn, we learn more about the available observations and the environment dynamics and refine our policy. We consider episodic environments, in which each episode ends either via a timeout (maximum number of steps is reached) or by reaching a goal state.

Interface. We formalize the setting via an interface comprising two operations through which the RL agent interacts with the environment: (1) **reset** and (2) **step**. Following the conventions of OpenAI gym [33], the **reset** operation resets the environment into its initial state. The **step** operation takes an action as input, performs the actions, which changes the environment state, and returns the immediate reward, a Boolean flag $done$, and the observation in the new state. The flag $done$ indicates whether a goal state was reached.

Execution & Traces. The agent learns in episodes, where it traverses a finite path in each episode. We want to note again that the agent cannot see the full state that it visited. Each executed path yields a finite reward-observation trace rt consisting of observations, immediate rewards, and the performed actions. We store these reward-observation traces in a multiset $\mathcal{RT}$.

7.3.2 Extended State Space

The Q-table in Q-learning is a function $Q : S \times I \to \mathbb{R}$, where S are the observable states of the environment. Since we only observe observations from a set O, we cannot use this function definition directly. As individual observations are insufficient to facilitate learning, we extend the observation space with states of a learned MDP leading to an extended state space. Suppose we are in episode i, we combine O with the states of the deterministically-labeled MDP $\mathcal{M}_i = \langle Q_i, I, O, \delta, \lambda_i, q_{0i} \rangle$, learned via IOALERGIA. During training, we continuously simulate the observation traces perceived by the RL agent on the learned automaton and use the visited states from S_i as additional observations. Since a learned MDP may not define transitions for all action-observation pairs, we represent the current state of $\mathcal{M}_i$ as a pair $(s, d) \in S_i \times \{\top, \bot\}$, where the first element encodes the last visited state of $\mathcal{M}_i$ and the second element denotes whether the simulation encountered an undefined transition. To work with these state pairs, we define two functions, where for $a \in I$ and $o \in O$:

$$resetToInitial() = (s_{0i}, \top)$$

$$stepTo((s,d), a, o) = \begin{cases} (s', \top) & \text{if } d = \top \wedge \delta_i(s,a)(s') > 0 \wedge O_i(s') = o \\ (s, \bot) & \text{otherwise} \end{cases}$$

The extended state space uses these state pairs, i.e., the Q-function is defined as $Q : S_i^e \times I \to \mathbb{R}$ with $S_i^e = O \times S_i \times \{\top, \bot\}$. Due to $\mathcal{M}_i$ being deterministic, s' in the above definition is either uniquely defined or not defined at all, denoted by $\bot$. Once we reach undefined behavior, we remember the last visited state and leave it unchanged. The intuition is that when behavior is encountered after

reaching some $(s, \perp)$ and it is important for RL performance, due to achieving high rewards, this will be reflected in updates of the Q-function. As a result, learning will be directed towards s. This leads to more sampling in the vicinity of s s.t. subsequently learned MDPs are more accurate in this region. Consequently, previously undefined behavior will eventually become defined in the learned MDP.

Note that in the current version of the algorithm, rewards are not included in the observation space, as the Q-learning serves to keep track of future rewards, whereas IOALERGIA keeps track of the structure of the environment. This separation of concerns helps to keep the size of learned MDPs small. The learned MDP states can be viewed as state estimators employed in other RL approaches in partially observable environments [40]. Carr et al. [40] rely on state estimation that yields belief supports, i.e., they estimate the currently possible environment states but abstract away the concrete state probabilities as they assume no knowledge about transition probabilities. In contrast to that, we learn transition probabilities and our discrete states include information on the probability of being in a particular environment state. However, these probabilities are only implicitly available to the RL agent, as the concrete states are not observable. To illustrate this, consider the BMDP shown in Figure 7.4. IOALERGIA may learn an MDP with the six states shown in the figure. Hence, the agent can distinguish the states reached after one, two, and three $coin$ inputs, respectively. A state estimator that solely estimates belief support would not be able to distinguish these states, since the beliefs $\mathbf{b}_{b1}$, $\mathbf{b}_{b2}$, and $\mathbf{b}_{b3}$ all have the same support.

7.3.3 Partially Observable Q-Learning

We apply tabular, ϵ-greedy Q-learning [269] combined with MDP learning. Deterministic labeled MDPs learned by IOALERGIA provide the Q-learning agent with additional information in order to make the learning problem Markovian despite partial observability.

We regularly learn new MDPs via IOALERGIA from the growing sample of reward-observation traces, where we discard the rewards, so that at each episode i there is an approximate MDP $\mathcal{M}_i$ with states S_i. To take information from $\mathcal{M}_i$ into account during RL, we extend the Q-table with observations corresponding to the states S_i. At every step performing action a and observing o during RL, we simulate the step in $\mathcal{M}_i$. This yields a unique state in S_i due to $\mathcal{M}_i$ being deterministic, which we feed to the RL agent as an additional observation.

We actually perform two stages of learning. First, we perform Q-learning while regularly updating $\mathcal{M}_i$. In the second stage, we fix the final MDP $\mathcal{M}_i$, referred to *freezing* below, and perform Q-learning without learning new MDPs with IOALERGIA. We term the resulting learning approach Q^{A}-learning.

Algorithm 7.1 implements this learning approach, i.e., training of a Q^{A}-learning agent. The implementation of the proposed algorithm, as well the code required to perform the empirical evaluation from Section 7.4 can be found online [1].

Algorithm 7.1 assumes that the Q^{A}-learning agent interacts with the environment *env* as described in Section. 7.3.1. The parameter *maxEp* defines the maximum number of training episodes. The other parameter *updateInterval* defines how often the agent recomputes the model, thus extending the state space perceived by the learning agent and the Q-table.

Lines 1-2 initialize the extended state space and set the actions of the agent to those of the environment. For the state-space initialization, we assume an initial approximate MDP to be given. In our implementation, we learn such an MDP from a small number of randomly generated traces. Alternatively, the extended state space S_E can be initialized to the observation space of the environment. It will in any case be extended as the algorithm progresses. Line 3 initializes the Q-table with the initial observation space and action space.

Training progresses until the maximum number of episodes is reached. In the implementation, we have added an early stopping criterion to end the training as soon as the agent achieves satisfactory performance on a predefined number of test episodes. Lines 5-17 show the steps taken in a single

[1] https://github.com/DES-Lab/Q-learning-under-Partial-Observability

Algorithm 7.1 Algorithm implementing Q^A-learning

Input: reinforcement learning environment env, fully configured partially observable agent $agent$, update interval $updateInterval$, number of training episodes $maxEp$

Output: trained $agent$ implementing the policy for the env

1: $agent.S_E \leftarrow env.O \times agent.model.states \times \{\top, \bot\}$ ▷ Init. extended state space
2: $agent.A \leftarrow env.A$ ▷ Get action state space from env
3: $agent.Q(s, a) = 0, \forall s \in S_E, a \in A$ ▷ Initialize the Q-table
4: $\mathcal{RT} \leftarrow \{\}$ ▷ Multiset of traces
5: **for** $trainingEpisode \leftarrow 0$ **to** $maxExp$ **do**
6: $initialObs, initialRew \leftarrow env.reset()$
7: $rt \leftarrow \langle initialObs, initialRew \rangle$ ▷ Initialize a trace of a single episode
8: $agentState \leftarrow agent.model.resetToInitial()$
9: $epDone \leftarrow False$
10: **while not** $epDone$ **do**
11: ▷ Select an action using ϵ-greedy policy and extended state space
12: $act \leftarrow agent.getAction(agent.state)$
13: ▷ Record all observed (state, action,reward, newState) pairs
14: $obs, reward, done, newObs \leftarrow env.step(act)$
15: $agentState \leftarrow agent.model.stepTo(agentState, act, newObs)$
16: $updateQValues(agent, obs, act, reward, newObs)$
17: $rt \leftarrow rt \cdot \langle act, reward, newObs \rangle$
18: $\mathcal{RT} \leftarrow \mathcal{RT} \uplus rt$
19: **if** $trainingEpisode \geq agent.freezeAutomaton$ **then** ▷ Freeze automaton
20: **continue**
21: **if** $trainingEpisode$ **mod** $updateInterval = 0$ **then**
22: $agent.model \leftarrow runIOAlergia(\mathcal{RT})$ ▷ Learn the new environment model
23: $agent.S_E \leftarrow agent.S_{Init} \times agent.model.states \times \{\top, \bot\}$
24: $agent.Q(s, a) = 0, \forall s \in S_E, a \in A$ ▷ Reinitialize the extended Q-table
25: **for** *episode* $\in \mathcal{RT}$ **do**
26: $agentState \leftarrow$ *agent.model.resetToInitial()*
27: **for** *obs, action, reward, newObs* $\in$ *episode* **do**
28: $agentState \leftarrow agent.model.stepTo(agentState, act, newObs)$
29: $updateQValues(agent, obs, action, reward, newObs)$
30: **return** $agent$

training episode. At the beginning of each episode, the environment and the agent's internal state are reset to their initial states (Lines 6 and 8). Until an episode terminates, either by reaching a goal or exceeding the maximum number of allowed steps, the Q^A-learning agent selects an action using an ϵ-greedy policy and executes it in the environment (Lines 12-14). Based on the selected action *act* and received observation *newObs*, the agent updates its current model state by tracing the pair *(act, newObs)* in the learned MDP (Line 15). After performing a step, the agent updates the values in the Q-tables based on observations and received reward. Algorithm 7.2 describes the process of updating Q-values. It follows the same procedure as in standard Q-learning with the notable difference of using a state space extended with learned MDP states instead of the observation space of the environment. The extended state space is discussed in more detail in Section 7.3.2

In Line 19, we check whether the automaton should be *frozen*. Freezing of the automaton prevents further updates of the model and extensions of the state space. This way once the automaton is frozen, the Q-table will continue to be optimized with respect to the current extended state space. Automaton freezing operates under the assumption that once a model is computed that is "good enough", computing a new model in the next update interval is unnecessary and might even be detrimental to the performance

Algorithm 7.2 Algorithm implementing update of Q-values of the agent

Input: Q^A-learning *agent*, environment *state*, *reward*, reached environment *newState*

1: $extendedNewState \leftarrow agent.model.stepTo(action, newState)$
2: $oldValue \leftarrow agent.Q(extendedState, action)$
3: $maxNextStateValue \leftarrow max(agent.Q(extendedNewState))$
4: $agent.Q(extendedState, action) \leftarrow (1-\alpha)*oldValue+\alpha*(reward+\gamma*maxNextStateValue)$

Table 7.1: Non-extended Q-table

State\Action	Up	Down	Left	Right
Room1	-0.35	-0.35	-0.45	-0.39
Room2	-0.67	-0.67	-0.32	-0.66
Room3	4.68	4.65	5.21	5.25
Room4	24.72	24.45	23.26	24.79

Table 7.2: Extended Q-table

State\Action	Up	Down	Left	Right
(Room1, s0)	-0.35	-0.35	0.75	-0.39
(Room1, s1)	-0.33	-0.35	0.90	-0.36
(Room1, s2)	-0.31	-0.31	1.06	-0.30
(Room1, s3)	0.4	-0.41	0.06	-0.35
...				
(Room2, s0)	0	0	0	0
(Room2, s5)	-0.67	1.23	1.07	-0.66
(Room2, s6)	-0.67	1.42	0.4	-0.66
...				
(Room4, s10)	97.3	74.1	78.2	93.8
(Room4, s11)	97.9	74.7	78.8	92.1
(Room5, s12)	98.3	75.2	79.2	95.3
...				

of the agent (in the short term).

If the automaton freezing is not enabled or its episode threshold has not yet been reached, we proceed with the update of the model and the Q-table (Lines 22-24). This update happens every *updateInterval* episodes. IOALERGIA computes a new model that conforms to the sample $\mathcal{RT}$ with rewards discarded. In Line 23 we extend the state space with state identifiers of the learned model. After that, we recompute the Q-table by initializing it with the extended state space and action space (Line 24). To recompute the values in the extended Q-table we perform an experience replay [79] with all traces in $\mathcal{RT}$ (Lines 25-29).

Example.

We use a simple *OfficeWorld* example to demonstrate state extension. In this example, an agent selects one of four actions: {*up, down, left, right*} to move into the given direction. The agent may also slip into a different direction with a location-specific probability.

Whereas Icarte et al.[104] and Neider et al. [178] use *OfficeWorld* in a non-Markovian reward setting under full observability, we modify the *OfficeWorld* layout as shown in Figure 7.1 to introduce partial observability. On the right-hand side of Figure 7.1, abstraction is applied over the state space. The RL agent can only observe in which room he is located, but not the x and y coordinates, which would truly identify a Markov state. Note that each observation, e.g. *Room1*, is shared by nine different MDP states identified by their x, y coordinates, each with different future and stochastic behavior.

Table 7.1 shows a Q-table obtained from observations only. The Q-values indicate that the Q-learning agent is unable to find optimal actions due to partial observability. We observe that for each observation in the non-extended Q-table, there is no clearly defined action that an agent must take in the current state to maximize the future reward, that is, the computed policy failed to identify a mapping from observations to actions that will consistently lead to a goal. Table 7.2 shows an extended Q-table, but we do not include the definedness flag from $\{\top, \bot\}$ for brevity. Each observation is extended with a learned

MDP state as discussed in Section 7.3.2. We observe that each *(observation, state)* pair approximates the underlying BMDP to such an extend that every such pair has a clear optimal action defined by the Q-values. For example, in states *(Room1, s0)* and *(Room1, s1)* the agent needs to perform a *left* action, whereas in state *(Room1, s3)* the agent needs to move *up*. We can also observe that the Q-values of the state *(Room2, s0)* are set to zero. This results from *s0* being unreachable while observing *Room2*.

7.3.4 Convergence

Q^A-learning learns an optimal policy in the limit, when the number of episodes tends to infinity, if the BMDP of the POMDP environment is finite, but when the BMDP is not finite and for finite sample sizes, we cannot guarantee convergence.

Finite BMDPs. Convergence for finite BMDPs follows from the convergence of IOALERGIA and Q-learning. We sample traces from a POMDP that, by assumption, is equivalent to a finite BMDP. The BMDP itself is a deterministic labeled MDP. IOALERGIA in the limit learns an MDP isomorphic to the canonical deterministic labeled MDP producing the traces when every action always has a non-zero probability to be executed, as has been shown by Mao et al. [144]. That is, every pair of belief-state and action would be explored infinitely often in the limit. This also ensures convergence of Q-learning in a Markovian environment [269]. The environment is Markovian once we learned the BMDP and add its current state to the observations of the Q-learning agent. Hence, we will learn an optimal policy for the BMDP and thus the POMDP in the limit.

Infinite BMDPs. We learn a finite-state approximation of the infinite BMDP. For instance, in the example shown in Figure 7.4, we might learn the three belief states labeled with $beep$, but beyond that the probability of observing tea is likely too small to detect additional states. As we will demonstrate in the evaluation, such approximate MDPs encode sufficient information to aid reinforcement learning. The learned automaton and its states can be thought of as providing memory to the reinforcement learning agent.

7.3.5 Limitations

The most notable limitation of the proposed approach is that, in its current form, it is only feasible with discrete observation space and action space. Continuous observation spaces would make IOALERGIA infeasible as it would fail to learn meaningful models (without the number of actions/observations approaching infinity) due to the high cardinality of the output set. Current state-of-the-art automata learning algorithms cannot cope with continuous, or more generally high-dimensional inputs and output spaces. In such instances, it is advisable to use recurrent neural networks, most notably LSTMs to capture state-space information [92]. While LSTMs can encode state information for continuous input and output spaces, they require substantially more data than automata learning approaches to achieve the same level of accuracy on discrete input/output observation sets.

7.4 Evaluation

To evaluate the proposed method we have implemented Q^A-learning in Python. The implementation uses AALPY's [166] IOALERGIA implementation and it interfaces OpenAI *gym* [33]. The implementation can be used on all *gym* environments with discrete action and observation space. We have evaluated Q^A-learning by comparing its performance on four partially observable environments with multiple state-of-the-art RL algorithms implemented in OpenAI's stable-baselines [94], considering: *non-recurrent policies with a stacked history of observations* and *LSTM-based policies*.

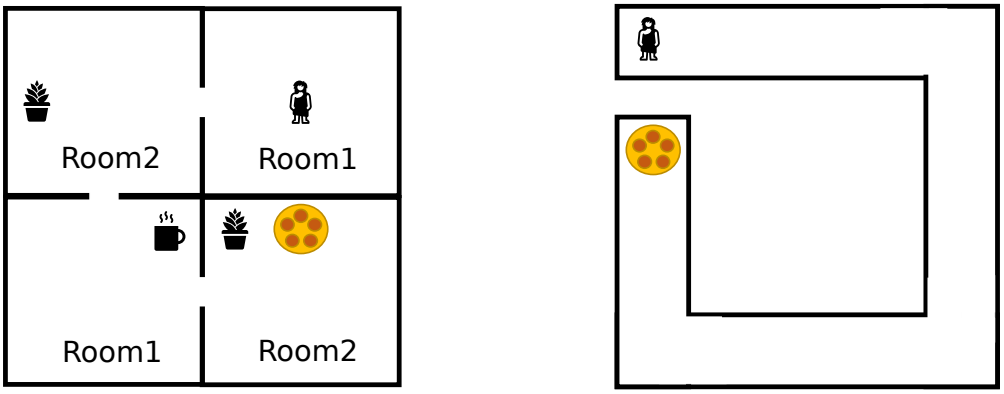


Figure 7.5: *ConfusingOfficeWorld* (left) and partially observable *ThinMaze* (right)

7.4.1 Competing Approaches

Stacked history of observations. In a first set of experiments, we have compared Q$^{\text{A}}$-learning with DQN [155], A2C [155], and ACKTR [273]. To aid those algorithms to cope with partial observablility, we encoded the history of observations as a frame stack. Stacked observation frames encode an observation history by using the last n observations observed during training and evaluation. By using stacked observations, we extend the observation space from initially i observations to $(i+1)^n$ observations [2]. For all experiments we have set the size of the frame stack to 5. A similar approach was, e.g., used in [155] as a method to encode movement in ATARI games.

Stacked history of observation-action pairs. To further help the previously mentioned algorithms to cope with partial observability, we also performed a set of experiments where the history stack was extended not only by the previous n observations but with the last n observation-action pairs, as done in [155]. This provides the RL algorithms with more state information, as the state explicitly encodes the actions that led to previously observed observations.

LSTM-based policies. Deep-recurrent Q-learning [92] has been used to solve ATARI games without stacking the history of observations. Hausknecht and Stone [92] show that recurrence is a viable alternative to frame stacking, and while no significant advantages were noticed during training of the agent, LSTM-based policies were more adaptable in the evaluation phase in the presence of previously unseen observations.

7.4.2 Setup of the Experimental Evaluation

All experiments were conducted on a laptop with an Intel® Core™i7-11800H at 2.3 GHz, 32 GB RAM, and an NVIDIA RTX™3050 Ti graphics card using Python3.6. For all experiments we have set the maximum number of training episodes to 30,000 for all environments, except for *ThinMaze* where it was set to 50,000. A training episode ends if an agent reaches a goal or the maximum number of steps is exceeded. The training performance was periodically evaluated and training was halted when reaching satisfactory performance.

Hyperparameters. Q$^{\text{A}}$-learning combines IOALERGIA and Q-learning. The sole parameter of IOALERGIA is the statistical significance value ϵ_{AL}, used for the state compatibility check, which we kept at 0.05. This is the default value presented in [144], and based on our experimentation different ϵ_{AL} do not significantly impact the model learning outcome. In Q-learning, the learning rate α is set to 0.1, while the discount factor γ is set to 0.9. The exploration rate ϵ linearly decreased during the algorithm execution, to provide more exploration at the beginning of the training. While the previous hyperparameters were intrinsic to Q$^{\text{A}}$-learning building blocks, we also define the freezing point and model update interval

[2] +1 due to the padding observation present in the first n steps of each training episode

Table 7.3: Representative evaluation results of Q^A-learning and competing approaches

Algorithm		OfficeWorld		Confusing OfficeWorld		GravityDomain		ThinMaze	
		# Steps to Goal	# Training Episodes	# Steps to Goal	# Training Episodes	# Steps to Goal	# Training Episodes	# Steps to Goal	# Training Episodes
Optimal Solution		12	-	12	-	18	-	20	-
Q^A-learning		12	3k	18	16k	18	3k	32	27k
Stacked observation	DQN	17	2k	✗	30k	75	30k	✗	50k
	A2C	24	12k	✗	30k	75	30k	✗	50k
	ACKTR	14	2k	✗	30k	75	30k	✗	30k
Stacked action observation	DQN	19	2k	12	3k	75	30k	✗	50k
	A2C	✗	30k	✗	30k	75	30k	✗	50k
	ACKTR	17	4k	✗	30k	✗	30k	✗	50k
LSTM-based policy	ACER	12	1k	✗	30k	75	30k	✗	50k
	A2C	12	1k	✗	30k	75	30k	✗	50k
	ACKTR	12	1k	✗	30k	75	30k	✗	50k

for Q^A-learning. The model update interval, that is, the number of episodes after which a new environment model is computed with IOALERGIA was set to 1000 episodes. The freezing point was set on an environment basis, usually after 10,000 episodes. The freezing point is currently a hyperparameter, while in future work, we will develop heuristics for dynamic freezing, for example, freezing based on the dissimilarity of two consecutive learned models. For more details about the hyperparameter configuration, we refer to the official implementation.

7.4.3 Results

Table 7.3 summarizes the results of the experiments. There are columns for each partially observable environment, where the first shows the average number of actions required to reach a dedicated goal with the best policy found by RL, and the second column shows the number of training episodes needed to learn a policy. The symbol ✗ denotes that no policy was found that reaches the goal within the allotted maximum number of steps. The rows correspond to: the optimal policy, the policy found by Q^A-learning, the three RL approaches with stacked observations, and the three approaches with LSTM-based policies. All experiments were repeated multiple times and we chose the best training run as a representative for each approach. As the agent performance was evaluated on 100 episodes, the average number of steps to reach a goal was rounded to the closest integer. In the remainder of this section, we will explain the partially observable environments on which the agents were trained and discuss the obtained results.

The *OfficeWorld* domain is depicted on the right of Fig. 7.1. Q^A-learning was able to find an optimal policy in this environment, but with a higher total number of training episodes compared to LSTM-based approaches. Stacked-frame based approaches also performed well, but were not able to find an optimal policy. This environment was solvable by all approaches despite its partial observability as each room has two actions which when executed repeatedly will lead the agent into the next room (e.g., in Room2 the agent needs to repeatedly perform down and right actions).

ConfusingOfficeWorld found on the left-hand side of Fig. 7.5 is a variation of the *OfficeWorld*. *ConfusingOfficeWorld* is harder to solve as the agent receives the same observations in the upper right and the lower left room, likewise in the upper left and the lower right rooms. The rooms labeled with *Room1* have two sets of opposite actions that need to be taken, depending on the actual agent location. The same holds for *Room2*. Q^A-learning was able to find a solution for this world in 16 thousand episodes, while the other approaches except for DQN with stacked action-observations pairs failed due to insufficient state differentiation.

GravityDomain was inspired by the environment discussed in [104]. In *GravityDomain*, gravity will pull the agent down in each state with 50% probability. By reaching a toggle indicated by a blue switch in Fig. 7.6, gravity is turned off and the environment becomes deterministic. We observed that both stacked-frame and LSTM-based approaches learned a policy, in which they repeatedly performed the

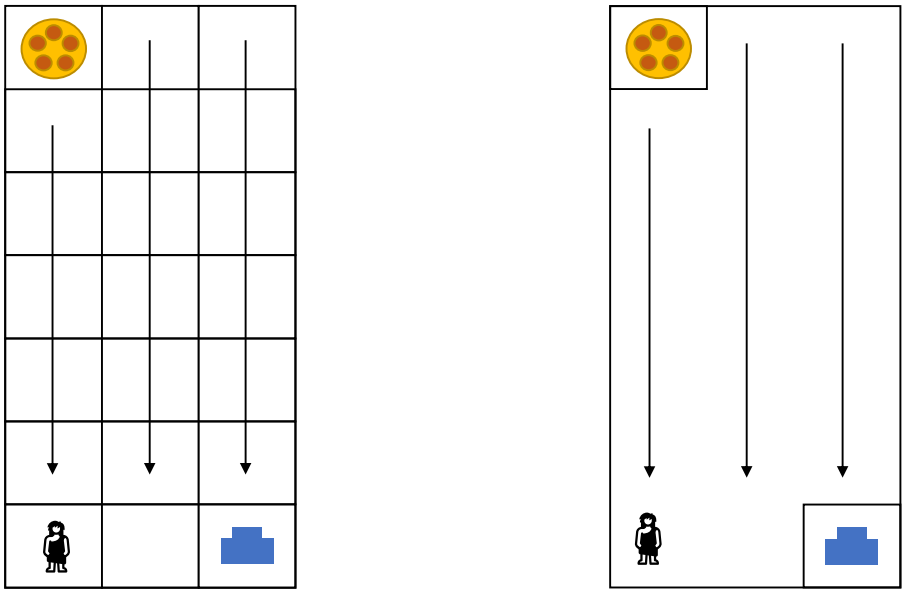


Figure 7.6: Fully and partially observable gravity domain. Once the button in the lower right corner is reached, gravity is turned off. Please note that for conciseness, we show a width of three, while we used six states in our evaluation.

up action, thus reaching the goal in only 50% of the test episodes within the maximum number of 100 steps. Q^A-learning was able to learn an optimal strategy in which it first reached the blue toggle and then proceeded to the goal, depicted by a cookie. Note that the approach presented in [104] is generally not able to solve the *GravityDomain*.

ThinMaze is depicted on the left-hand side of Fig. 7.5. In *ThinMaze*, the only observations are "cookie" and "wall", which signals that the agent performs an action that is blocked by a wall. Due to the lack of observations/state differentiation both stacked-frame and LSTM-based approaches failed to find a solution to *ThinMaze*. Q^A-learning was able to find a non-optimal solution in 27 thousand training episodes. This is due to the fact that IOALERGIA requires a high number of traces to approximate the underlying belief-MDP with sufficient accuracy.

Runtime and model size. We only briefly comment on runtime, considering *OfficeWorld* for a fair comparison, as all approaches found a decent policy. Other results might be skewed due to different training lengths. Stacked-frame DQN, A2C, and ACKTR, require $312s$, $373s$, and $48s$, respectively. Stopping after only $1k$ episodes, the LSTM-backed ACER, A2C, and ACKTR take $51s$, $74s$, and $80s$, respectively. Our approach is considerably faster, finishing after about $3s$. The reason is that we apply tabular Q-learning and IOALERGIA adds very little runtime overhead. The automata-learning technique has cubic worst-case runtime in the sample size, but has been reported to have linear runtime in practice [41]. Due to the small size and high degree of partial observability found in all used environments, IOALERGIA learns relatively small MDP-approximations of underlying POMDPs. However, those models provide enough information to the Q^A-learning so that it may solve the RL tasks in all environments. Over multiple experiments, Q^A-learning learned a 28-state model for the *OfficeWorld* and for the *ConfusingOfficeWorld*, a 15-state model for $Gravity$ and a 26-state model for the $ThinMaze$ environment.

Discussion about Scalability. All experiments were conducted in testing environments with a high degree of partial observability. To be more precise, in all considered environments each observation labeled many distinct neighboring Markov states that need to be distinguished by effective policies. Hence, we show that automata learning can effectively help to deal with incomplete information resulting from partial observability. To really focus on the aspect of partial observability, we limited experiments to

relatively small, yet challenging environments, where we compare to state-of-the-art deep RL baselines. We argue that our approach can scale to high-dimensional environments with the help of feature extraction methods, like "image segmentation algorithms" assumed by DeepSynth [91], that enable automata learning over abstract observations corresponding to extracted features. In such a setting, IOALERGIA over abstract observations could be combined with deep RL methods working in large unstructured state spaces. Learned MDPs would benefit RL by providing insights into the abstract structure of the problem domain.

Note that our general approach could also work even more efficiently in environments with less uncertainty. In deterministic environments, we could learn DFAs to identify the hidden state structure, e.g., via RPNI [187]. There we would actually require less data for accurate learning since statistical compatibility checks are not necessary. If an environment can be modeled by a *finite* deterministic labeled MDP, i.e., it has finitely many belief states, IOALERGIA would converge to the true model.

7.5 Concluding Remarks

We have shown how passive stochastic automata learning and Q-learning can be combined in order to enable RL in POMDPs. The proposed method works on environments with discrete inputs and outputs. However, many modern RL problems are defined in continuous stochastic environments. These RL problems are not suited for tabular methods such as Q-learning due to their large state-space and are often solved with neural networks. What is more, large continuous state-spaces are not suitable for automata learning algorithms. We tackle this challenge in the next chapter, where we present a method that is able to model agents' behavior in such continuous stochastic environments with the help of automata learning and dimensionality reduction and clustering.

RQ 3.1 Can automata learning help reinforcement learning?

In this chapter, we proposed an approach for reinforcement learning under partial observability guided by learning environment models. For this purpose, we combine Q-learning with IOALERGIA. With automata learning, we learn hidden information about the environment's state structure that provides additional observations to Q-learning, thus enabling this form of learning in POMDPs. We evaluate our approaches in partially observable environments and show that it can outperform the baseline deep RL approach with LSTMs and fixed memory.

8

AUTOMATA LEARNING MEETS SHIELDING

Declaration of Sources

The content presented in this chapter is based on our paper "Automata Learning Meets Shielding" which was presented at ISOLA2022 [241].

8.1 Motivation

In the previous chapter, we have shown how automata learning can help reinforcement learning in the presence of partial observability. This chapter addresses *safety* [120], another major challenge in reinforcement learning. More precisely, we will show how automata learning in combination with shielding can be used to enforce safety during the training process.

Shielding [12] is a runtime enforcement technique that applies correct-by-construction methods to automatically compute shields from a given safety temporal logic specification [19] and a model that captures all safety-relevant dynamics of the environment. Shields have been categorized into post-shields and pre-shields [120]. Post-shields monitor the actions selected by the agent and overwrite any unsafe action with a safe one. Pre-shields are positioned in front of the agent and block, at every time step, unsafe actions from the agent. That is, the agent can only choose from the set of safe actions. We use pre-shielding since it provides the agent with maximal freedom in the environment exploration during training, that is, the shield only restricts the set of available actions in each state, and is not used to select a safe action instead of an agent under training.

In the non-probabilistic setting [30], shields guarantee that the safety specification will never be violated, working under the assumption that a complete and faithful environmental model of the safety-relevant dynamics is available. Shielding in the probabilistic setting [112], which is the standard setting in RL, assumes to have an environmental model in the form of an MDP available. Given such an MDP $\mathcal{M}$ and a safety specification φ, the shield computes how likely it is that executing an action from the current state will result in violating φ within a given finite horizon. At any state s, an action a is called *unsafe* if executing a incurs a probability of violating φ within the next k steps greater than a relative threshold λ with respect to the optimal safety probability possible in s. The resulting shield prohibits safety violations that can be prevented by planning k steps into the future. Shielding requires a complete and accurate environmental model, but it is rarely the case that such a model is available. To solve this challenge we turn to automata learning.

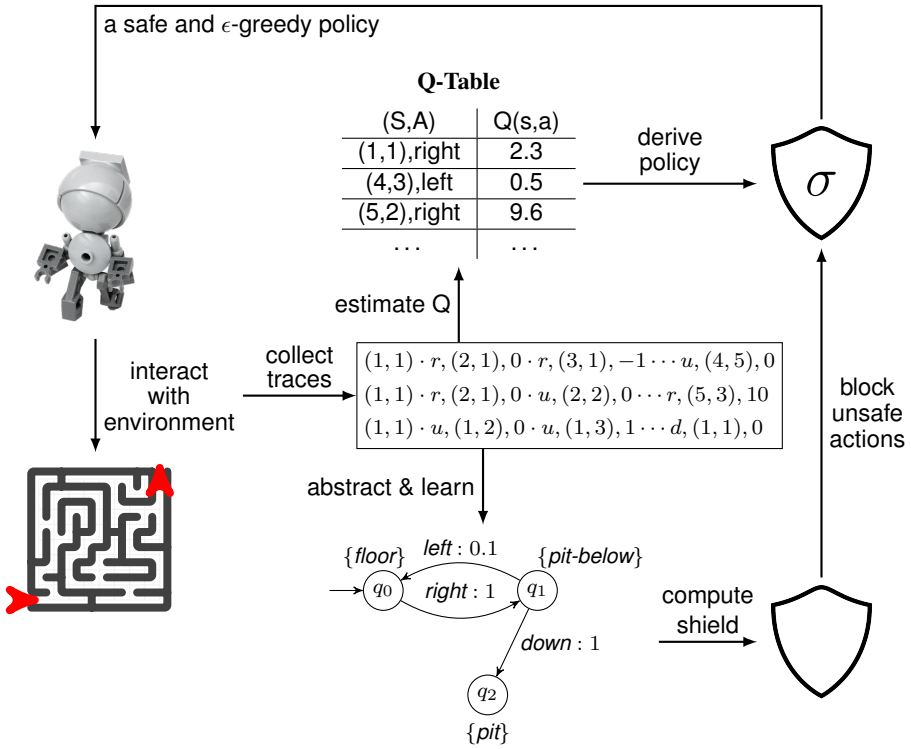


Figure 8.1: Iterative safe reinforcement learning via learned shields

8.1.1 Guiding Idea

The goal of the proposed approach is to reduce safety violations during the exploration phase of the RL training by combining passive MDP learning with probabilistic shielding in an iterative approach.

The idea behind our approach is abstractly presented in Figure 8.1. We assume no knowledge of the environment's structure and only assume the existence of an abstraction function that can abstract concrete observations of the environment to the abstract observations that are relevant to some safety specification.

During the training of the RL agent, the agent collects observation traces while exploring the environment. After having a large enough initial multiset of traces, we first transform the sequences of observed states in the traces into observations, which include only the safety-relevant information, using a suitable abstraction function. The abstract traces are then used to learn a first estimate of the safety-relevant MDP. From this initial MDP and a given safety specification, we construct an initial shield. After the shield is constructed, the agent is augmented with the shield, i.e., the shield blocks unsafe actions from the agent and the agent can pick from the set of safe actions. The newly collected traces are added to the multiset of all traces. After collecting a predefined number of new traces, our approach learns a new safety-relevant MDP from the multiset of all collected traces and creates a new shield. At every iteration, the shield is built from an MDP that approaches more and more the real MDP underlying the environment modulo the abstraction to safety-relevant observations. Thus, the resulting shields are getting more informed and prevent the agent from entering more safety-critical situations. Note that the learned model is computed only with abstract observations that are relevant to the safety of the agent, and it is only used to block unsafe actions during the training of the RL agent, and does not provide the agent with additional information as in the previous chapter.

The proposed approach shares similarities with Q^A-learning, which was presented in Chapter 7. Both approaches work on discrete environments and use learned models to help RL agents. In the previous chapter, we have used the learned model to extend the observation space available to the agent, while in

this chapter the learned abstract model is used in shield computation which blocks unsafe actions during RL training.

Structure. This chapter is structured as follows: in Section 8.2 we outline basic preliminaries on shielding and propose a method of safe RL training via shielding over learned models, which we then evaluate in Section 8.3. We conclude the chapter and position this work in the context of the thesis in Section 8.4.

8.2 Enabling Shielding Via Automata Learning

In this section, we present our iterative approach for safe reinforcement learning via automata learning and shielding. We start by outlining basic preliminaries on shielding, followed by a discussion on the setting in which the RL agents operates and give our problem statement. Then we give an overview of our approach. Finally, we discuss the individual steps of our approach in detail.

Preliminaries. We rely on standard definitions of MDPs, RL, and passive learning of MDPs presented in Chapter 2. Note that in the context of this chapter we use terms "MDP inputs" and "actions" interchangeably.

8.2.1 Brief Introduction to Shielding of MDPs

Specifications and Model Checking. We consider specifications given in the safety fragment of linear temporal logic (LTL) [19]. For an MDP $\mathcal{M}$ and a safety specification φ, probabilistic model checking employs linear programming or value iteration to compute the probabilities of all states and actions of $\mathcal{M}$ to satisfy a φ. Specifically, the probabilities $\eta^{\max}_{\varphi,\mathcal{M}} \colon Q \times \mathbb{N} \to [0,1]$ or $\eta^{\min}_{\varphi,\mathcal{M}} \colon Q \times \mathbb{N} \to [0,1]$ give for all states the maximal (or minimal) probability over all possible policies to satisfy φ, within a given number of steps. For instance, a safety property $\varphi = \mathbf{G}(\neg Q_{\textit{unsafe}})$ could encode that a set of unsafe states $Q_{\textit{unsafe}} \in Q$ must not be entered. Then $\eta^{\max}_{\varphi,\mathcal{M}}(s,h)$ is the maximal probability to not visit $Q_{\textit{unsafe}}$ from state $s \in Q$ in the next h steps.

Shield Construction. Given an MDP $\mathcal{M}$, a safety specification φ, and a finite horizon h, the task of the shield is to limit the probability of violating the safety specification φ within the next h steps.

For any state $s \in Q$ and action $a \in I$, the *safety-value* $val_{\varphi,\mathcal{M}}(s,a,h)$ is computed which gives the maximal probability to stay safe from s after executing a, i.e.,

$$val_{\varphi,\mathcal{M}}(s,a,h) = \eta^{\max}_{\varphi,\mathcal{M}}(\delta(s,a), h-1).$$

The *optimal safety-value* $optval_{\varphi,\mathcal{M}}(s)$ of s is the maximal safety value of any action a in state s within the next $h-1$ steps, i.e.,

$$optval_{\varphi,\mathcal{M}}(s,h) = \max_{a \in I(s)} val_{\varphi,\mathcal{M}}(s,a,h) = \eta^{\max}_{\varphi,\mathcal{M}}(\delta(s,a), h-1).$$

An action a in s is *unsafe* if the safety value of a is lower than the optimal safety-value by some threshold λ_{sh}, i.e., an action a in state s is *unsafe* iff

$$val_{\varphi,\mathcal{M}}(s,a,h) < \lambda_{sh} \cdot optval_{\varphi,\mathcal{M}}(s,h).$$

We refer to actions that are not unsafe as safe actions.

The task of the shield is to block any unsafe action from the agent, thereby restricting the set of available actions I to the set of safe actions. A shield is a relation $\pi_{\bigcirc} : Q \to 2^{I(s)}$ allowing at least one action for any state.

8.2.2 Overview

Setting. We consider an RL agent acting in an unknown environment that can be modeled as an MDP $\mathcal{M} = \langle Q, I, O, \delta, \lambda, q_0 \rangle$ with an associated reward function $R : Q \to \mathbb{R}$. However, since the environment is unknown at the beginning of the learning phase, the agent has no knowledge about the structure of $\mathcal{M}$.

We assume that the safety-critical properties are given in the form of an LTL formula φ. Without knowledge about the safety-relevant dynamics of the environment, it is not possible to prohibit violating φ while the RL agent is exploring the environment.

During the exploration phase of the RL agent, a multiset of reward traces is collected. We assume to have an observation function $Z : Q \to O$ given that maps any state $s \in Q$ states to a safety-relevant observation $o \in O$.

Problem Statement. Consider the setting as discussed above. The goal of our approach is to use the available data from the environment in the form of collected traces and the given safety specification to prevent safety violations if possible.

Our approach combines automata learning, shielding, and reinforcement learning in an iterative manner. Our approach performs n_{iter} iterations. At the first iteration with $i = 1$, we start from an empty multiset of reward traces $\mathcal{T} = \emptyset$, an initial learned $\mathcal{M}_{\bigcirc 0}$ with a single state and a self-loop for any action, and an initial shield $\pi_{\bigcirc 0}$ that allows at every step any action.

At each iteration i of n_{iter} iterations, our approach works as follows:

1. **Exploration**: The RL agent explores the environment to learn an optimal policy. At each iteration i, the agent trains for a number of $n_{episodes}$. The agent is augmented by a shield $\pi_{\bigcirc i-1}$ that restricts its available actions. The learned MDP $\mathcal{M}_{\bigcirc i-1}$ is simulated in parallel during the exploration. Each training episode yields a trace τ that is added to the multiset of all collected traces $\mathcal{T}$, i.e., $\mathcal{T} = \mathcal{T} \uplus \{\tau\}$.

2. **MDP Learning**: After executing $n_{episodes}$ episodes, we learn a deterministic labeled MDP $\mathcal{M}_{\bigcirc i}$ from $\mathcal{T}$.

3. **Shield Construction**: Using $\mathcal{M}_{\bigcirc i}$, a given specification φ, relative safety threshold λ_{sh}, and a finite horizon h, we compute a shield $\pi_{\bigcirc i}$ and continue in **Step 1: Exploration**.

Step 1. Exploration

The agent interacts with an unknown stochastic environment $\mathcal{M} = \langle Q, I, O, \delta, \lambda, q_0 \rangle$. The agent picks actions that are considered to be safe by the current shield and receives observations of the state of the environment and rewards. At iteration i, we are given the set of collected reward traces $\mathcal{T}$, the learned MDP $\mathcal{M}_{\bigcirc i-1} = \langle Q_{\bigcirc}, I_{\bigcirc}, O_{\bigcirc}, \delta_{\bigcirc}, \lambda_{\bigcirc}, q_{0\bigcirc} \rangle$, and the current shield $\pi_{\bigcirc i-1}$. Each episode resets the environment and starts in a fixed initial state $q_0 \in Q$ for $\mathcal{M}$ and $q_{\bigcirc 0} \in Q_{\bigcirc}$ for $\mathcal{M}_{\bigcirc}$. At every step t, the agent observes $q_t \in Q$. Based on the observed label $Z(q_t)$, the learned MDP $\mathcal{M}_{\bigcirc i-1}$ moves to state $q_{\bigcirc t}$. We give in Section 8.2.2 the details on how to simulate $\mathcal{M}_{\bigcirc i}$.

The shield determines the set of safe actions $I_{\bigcirc} = \pi_{\bigcirc i-1}(q_{\bigcirc t})$ and sends it to the agent, who then selects a safe action $a_{t+1} \in I_{\bigcirc}$. This is usually implemented by selecting the action with the largest Q-value from the set of safe actions. The environment executes a_{t+1} and sends the agent a reward r_{t+1} and state observation s_{t+1}. Based on $(s_t, a_{t+1}, r_{t+1}, s_{t+1})$, the agent performs a policy update. Figure 8.1 represents the learned Q-function of the RL agent as a Q-table, from which we derive an ϵ-greedy policy for training. Please note that our approach is general and applicable to deep Q-learning as well as to tabular Q-learning.

A training episode ends after a maximal number of t_{max} steps. It may end earlier in case of violating safety or completing the task that needs to be learned (e.g., reaching a certain goal state). Each episode yields a reward trace $\tau = s_0 a_1 r_1 s_1 \cdots s_{n-1} a_n r_n s_n$ that is added to the multiset $\mathcal{T}$ of all collected traces, i.e., $\mathcal{T} = \{\tau\} \uplus \mathcal{T}$.

Step 2. MDP Learning.

After $n_{episodes}$ training episodes, resulting in a new reward trace per episode, we learn a new model of the environment dynamics. Given the multiset of reward traces $\mathcal{T}$ observed while exploring the environment, we abstract away all information in the traces that is not relevant to safety. For each reward trace $\tau = s_0 a_1 r_1 s_1 \cdots s_{n-1} a_n r_n s_n \in \mathcal{T}$, we first discard the rewards to obtain a path $\rho = s_0 a_1 s_1 \cdots s_{n-1} a_n s_n$ and second apply the abstraction function Z to obtain an observation trace $\tau_o = Z(\rho) = Z(s_0) a_1 Z(s_1) \cdots Z(s_{n-1}) a_n Z(s_n)$. For example, the states in the path ρ may represent the exact coordinates of the agent's positions and distance measurements. Relevant for safety might only be the distances. In such a case, the abstraction function Z would abstract away the concrete position and only keep the distances.

The deterministic labeled MDPs $\mathcal{M}_{\bigcirc i}$ are learned with IOALERGIA. Additionally, we make $\mathcal{M}_{\bigcirc i}$ *input-complete*. During the training phase, we propose to make $\mathcal{M}_{\bigcirc i}$ with $i < n_{iter}$ input-complete by adding self-loop transitions to all state-action pairs $(s_{\bigcirc}, a_{\bigcirc})$ where $\delta_{\bigcirc}(s_{\bigcirc}, a_{\bigcirc})$ is not defined. That is, we set for all such state-action pairs $\delta_{\bigcirc}(s_{\bigcirc}, a_{\bigcirc}) = \{s_{\bigcirc} \mapsto 1\}$. The rationale behind this is that whenever an action's effect is unknown, we assume that it leaves the safety of the corresponding state unchanged.

For the final shield $\pi_{\bigcirc}$ that will be used permanently after training, we propose a more conservative approach and to make $\mathcal{M}_{\bigcirc i}$ with $i = n_{iter}$ input-complete by adding a special sink state $s_{\neg\varphi}$ that violates φ and adding transitions to $s_{\neg\varphi}$. In the training phase, the resulting shield $\pi_{\bigcirc}$ would not be suitable since it would prohibit exploration. For the final deployment phase, however, the behavior of $\pi_{\bigcirc}$ may be desirable since it blocks behavior that has not been explored sufficiently.

Step 3. Shield Construction

The abstract MDP $\mathcal{M}_{\bigcirc i}$ encodes the safety-relevant information about the environment. We use $\mathcal{M}_{\bigcirc i}$, the safety specification φ, and a finite horizon h, to compute the safety values of all state-action pairs in $\mathcal{M}_{\bigcirc i}$. Based on a given relative threshold λ_{sh}, we compute a shield $\pi_{\bigcirc i}$ which allows for any state all actions that are safe with respect to λ_{sh} and the optimal safety value. After constructing the shield $\pi_{\bigcirc i}$, the current shield of the agent is set to $\pi_{\bigcirc i}$. The agent continues to explore the environment and learn the optimal strategy using $\pi_{\bigcirc i}$ (Step 1). To compute shields, we use TEMPEST [199], an open-source synthesis tool that can be used to compute shields in probabilistic environments.

Details for Training with Learned Shields

In this section we discuss the execution of a single RL episode of an agent augmented with a learned shield, which we show in Algorithm 8.1. The RL agent interacts with the environment through two operations: $reset$ and $step$, where $reset$ is used to bring the environment to a fixed initial state, and $step$ is used to perform a single step on the environment, that is, it takes an action a and executes a causing a probabilistic state transition. It returns a pair r, s', where r is the reward gained by reaching s' and s' is the reached state when executing a.

The algorithm starts with initializing the learned MDP state as well as the environment state (Line 1 and 2). Then we enter a loop in which the agent performs a maximum of t_{max} steps. The RL agent applies ϵ-greedy learning, i.e., it explores a random action with probability ϵ and otherwise performs the optimal action according to the RL agent's current knowledge. Note that both, random exploration and exploitation, are shielded, i.e., actions are chosen from the set of safe actions. For every step, the shield provides a set of safe actions (Line 5). With probability ϵ, the RL agent selects a random safe action in Line 7. Otherwise, it determines the currently optimal (safe) action in Line 9. The chosen action is executed in Line 10. In Line 11, the state of $\mathcal{M}_{\bigcirc}$ is updated. To conclude the training step, we update the agent's Q-table. If the agent visits a terminal state, the loop terminates before performing t_{max} steps (Line 13). A terminal state may, for instance, be reached by completing the task to be solved or by violating safety. After training, we use the same approach to execute an agent, but with ϵ set to 0, such that only safe and optimal actions are executed.

Algorithm 8.1 A single RL training's episode using a learned shield

Input: Q-function Q, a learned deterministic labeled MDP $\mathcal{M}_{\bigcirc} = \langle Q_{\bigcirc}, I_{\bigcirc}, O_{\bigcirc}, \delta_{\bigcirc}, \lambda_{\bigcirc}, q_{0\bigcirc} \rangle$, exploration rate ϵ, safety shield $\pi_{\bigcirc}$, environment with *step* and *reset*

Output: Updated Q-function Q

1: $s_{\bigcirc} \leftarrow q_{0\bigcirc}$
2: $s \leftarrow$ **reset**
3: $t \leftarrow 0$ ▷ step counter
4: **while** $t < t_{max}$ **do**
5: $I_{\bigcirc} \leftarrow \pi_{\bigcirc}(s_{\bigcirc})$ ▷ safe actions for s
6: **if** *coinFlip*(ϵ) **then**
7: $a \leftarrow randSel(I_{\bigcirc})$
8: **else**
9: $a \leftarrow \text{argmax}_{a' \in I_{\bigcirc}} Q(s, a')$
10: $r, s' \leftarrow \text{STEP}(a)$ ▷ step in environment
11: $s_{\bigcirc} \leftarrow s'_{\bigcirc}$ where $\delta(s_{\bigcirc}, a)(s'_{\bigcirc}) > 0$ and $\lambda(s'_{\bigcirc}) = Z(s')$ ▷ step in learned MDP
12: $\text{UPDATEQ}(s, a, r, s')$ ▷ Update Q-values
13: **if** s' is a terminal state **then**
14: **break**
15: $t \leftarrow t + 1$

8.3 Evaluation

In our experimental evaluation, we evaluated our approach on 24 slippery gridworld environments of varying shapes and sizes. We implemented a tabular Q-learning agent that should learn to reach a given goal state quickly and safely, that is, by minimally violating the safety specification. We learn deterministic labeled MDPs using IOALERGIA implemented in AALPY [166]. The shields are created from the learned MDPs using the shield synthesis tool TEMPEST [199]. We discuss the scalability of our approach and its effectiveness by comparing the average gained reward during training with and without learned shields.

All experiments have been executed on a desktop computer with a 4 x 2.70 GHz Intel Core i7-7500U CPU, 8 GB of RAM running Arch Linux. An implementation of the proposed method and evaluation framework are available online[1].

8.3.1 Case Study Subjects

Gridworlds. We used three types of parameterized gridworlds, the smallest instances of which are depicted in Figure 8.2, Figure 8.3, and Figure 8.4. Each gridworld has a dedicated *start* tile and a *goal* tile, marked by *Entry* and *Goal* on the left-hand side of the figures. A gridworld might also have *intermediate-goal* tiles. If a tile is marked as *pit*, then this tile marks an unsafe location and the agent is not allowed to visit this tile, that is, visiting of this tile terminates the episodes and incurs a safety violation. Black tiles represent *walls* that restrict movement but are not safety-critical. Additionally, each tile has a *tile abstraction*, which we use to simplify the presentation of the abstraction function, denoted by lowercase letters on the right-hand side of the figures. If a tile is "slippery", the tile is labeled with an arrow and grey gradients on the left-hand side of the figures. If an agent tries to move from a slippery tile, the intended movement of the agent might be altered into the direction of the arrow with a specific tile-dependent probability. The length of arrows corresponds to the probability of slipping. That is, a long arrow pointing downward left means that the agent is very likely to slip either to the left or

[1] https://github.com/DES-Lab/Automata-Learning-meets-Shielding

Figure 8.2: The smallest of the zigzag gridworlds

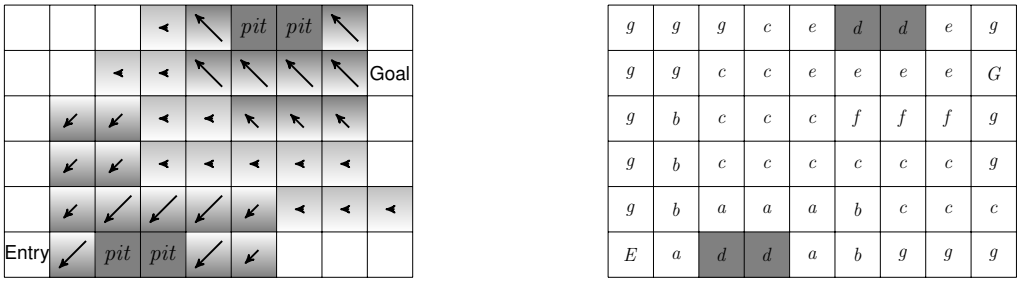


Figure 8.3: The smallest of the slippery shortcut gridworlds

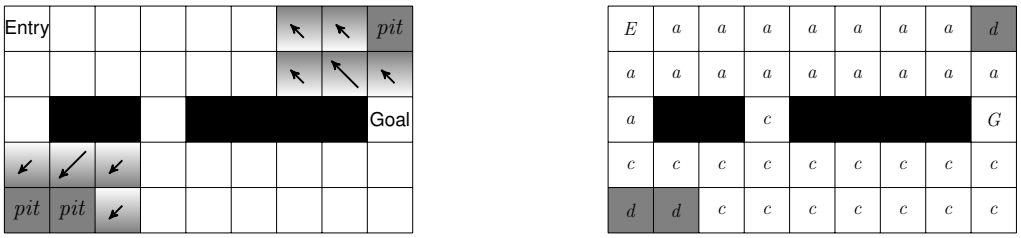


Figure 8.4: The smallest of the wall gridworlds

downwards. Additionally, since environments are fully observable each tile has (x, y) coordinates that are available to the agent.

We discuss each of the three gridworld shapes briefly, where Figures 8.2 to 8.4 show the smallest gridworld of each shape. To get insights into how the state space affects learning, we vary each type of gridworld in size by creating eight versions of increasing size.

- *zigzag*: In the *zigzag* gridworlds illustrated in Figure 8.2, we have pairs of pit tiles located in alternation at the bottom and the top of the map with a fixed distance between adjacent pit pairs. The goal is placed so that the agent could move along a straight line from left to right, but it would travel across slippery tiles next to the pits. The shield thus helps to avoid these dangerous tiles to perform zigzag walks to the goal.
- *slippery shortcuts*: The *slippery shortcuts* gridworlds illustrated in Figure 8.3 are similar to the *zigzag* maps in that the agent could take shortcuts across slippery tiles next to pits. In this case, the probability of unsuccessful moves decreases with increasing distance from the pits. Hence, an optimal policy has to find a balance between taking risks and gaining higher rewards due to shorter paths.
- *walls*: In the *walls* gridworlds illustrated in Figure 8.4, the agents must find a way from the start to the goal by navigating around walls and pits that block the shortest paths. There are fewer slippery tiles.

The following reward structure was used for all gridworlds:

- If the agent reaches the goal within less than t_{max} steps, it receives a reward of $+100$. Additionally, reaching the goal ends the training episode.
- If the agent reaches a tile with a *pit*, it receives a punishment of -100 and the training episode ends.
- Additionally, the agent receives a reward of -0.5 per step, which penalizes policies that require many steps to reach a solution.

- Gridworlds might have intermediate goal states that are rewarded with $+20$ but do not terminate the episode.

8.3.2 Experimental Setup

Reinforcement Learning. We implemented a tabular Q-learning agent. The agent's task is to navigate from start to goal by moving into one of the four directions at each time step.

Every training episode, the agent collects reward traces of the form $\tau = (x_0, y_0), a_1, r_1, (x_1, y_1) \dots a_n, r_n, (x_n, y_n)$ with x_i and y_i representing the x and y coordinates of the tiles. The set of available *actions* comprises of the four actions {left, right, up, down} to move into the corresponding direction.

We use the following parameters for all experiments: The tabular Q-learning agent was trained with a learning rate of $\alpha = 0.1$, a discount factor of $\gamma = 0.9$, and an initial exploration rate of $\epsilon = 0.4$ throughout all experiments. We chose an exponential epsilon decay of $\epsilon' = 0.9999 \cdot \epsilon$ with an update after every learning episode. For IOALERGIA, we used the default ϵ_{AL} value of 0.05.

Safety-relevant Abstraction Function. To get the observations for MDP learning, we perform an abstraction over the states via the function Z. Given a concrete state (x, y), the function $Z((x, y))$ maps to a pair (ab, Pit), where ab is the abstract representation of the tile at (x, y) and Pit is a set of propositions denoting whether a pit is located in the neighboring tiles in each of the four cardinal directions.

For example, if the agent is at the coordinate $(1, 0)$ in Figure 8.2, with the origin of coordinates on the bottom left, the abstract observation would be $Z((1, 0)) = (c, \{\textit{pit-right}\})$. The abstraction function maps to c and there is a pit on the right.

Shield Construction. In all experiments, visiting a pit represents a safety violation. This property can be represented in LTL as follows: $\varphi = \mathbf{G}(\neg pit)$. Using this specification φ and a deterministic labeled MDP $\mathcal{M}_{\bigcirc}$, we compute a shield $\pi_{\bigcirc}$ using a relative threshold $\lambda_{sh} = 0.95$ and a finite horizon of $h = 2$. Thus, for any given state, the resulting shield $\pi_{\bigcirc}$ allows an action a if the probability of not falling into a pit within the next 2 steps is at least $0.95 \cdot \alpha$, with α being the probability of not falling into a pit when taking the optimal action a'.

8.3.3 Experimental Results

In this subsection, we report on the performance of RL agents augmented with learned shields compared to the performance of unshielded RL agents.

To account for the stochastic nature of the environment and RL, we repeat every experiment 30 times. For each experiment, the number of iterations is set to $n_{iter} = 30$, the number of training episodes per iteration is $n_{episodes} = 1000$, and the maximal length of a training episode is set to $t_{max} = 200$ steps.

To evaluate the learned policies, we execute the policies at various stages throughout training and compare their performance. For every iteration, i.e., after every 1000^{th} training episode, we evaluate the intermediate policies of the agents by setting the exploration rate ϵ to 0 and performing 1000 episodes. Over these 1000 episodes, we compute the average cumulative reward of the intermediate policy. We refer to this value as *return*. For shielded agents, their corresponding shields are used during the intermediate executions for evaluation.

Figure 8.5, Figure 8.7, and Figure 8.9 show plots of the results of this evaluation, where the x-axes display the episodes and the y-axes display the return. The average performance of the *shielded agents* is represented by a thick *green line*, whereas a thick *red* line represents the average performance of *unshielded agents*. The light green and light red areas depict the range between the minimum and maximum performance of the shielded and the unshielded agents, respectively. Figure 8.6, Figure 8.8, and Figure 8.10 depict the number of safety violations throughout training, where the x-axes display the episodes and the y-axes display the number of times the agent visited a pit. The average number of safety violations of the *shielded agents* are represented by a thick *green line*, whereas a thick *red* line represents the average number of safety violations of *unshielded agents*. The light green and light red

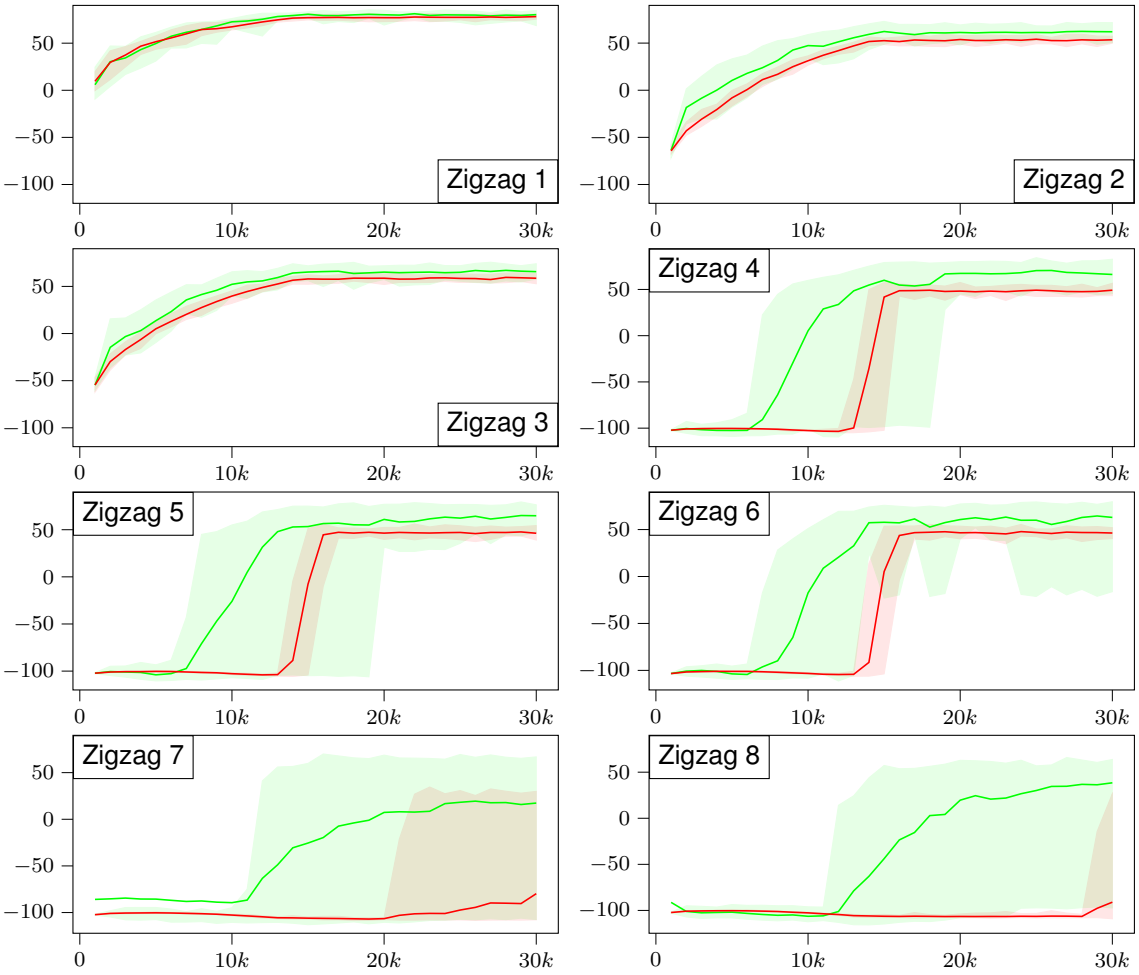


Figure 8.5: The return gained by intermediate policies throughout reinforcement learning in the *zigzag* gridworlds. The y-axes display the return and the x-axes display the episodes at which policies are evaluated. The green plots represent shielded performance, whereas the red plots represent unshielded performance.

areas depict the range between the minimum and maximum number of safety violations of the shielded and the unshielded agents, respectively. In the following, we discuss the results from the experiments with the different gridworld shapes.

8.3.4 Zigzag Gridworlds

We start by discussing the performance of the RL agents in the *zigzag* gridworlds. Figure 8.5 shows the return, and Figure 8.6 shows the number of safety violations at various stages of the RL training.

The experiments in Figure 8.5 show almost identical returns and number of safety violations for shielded and unshielded agents for the three smallest gridworld instances. Starting with the fourth-smallest gridworld, shielded RL performs better on average. Initially, during the first episodes of both RL configurations, the average return is approximately -100, which is the penalty for falling into a pit. This means that the agent consistently falls into pits in the early stages of learning. After approximately 7 iterations, i.e, at episode 7000, the shielded agents start to reach the goal states, which leads to an increase in the return and a decrease in the number of safety violations. Unshielded agents need about twice the training time to reach the goal location. Similar observations can be made for the next larger environments *Zigzag 5* and *Zigzag 6*, too. For the two largest *zigzag* gridworlds, unshielded RL fails to consistently reach the goal after $30,000$ training episodes. These two environments require rel-

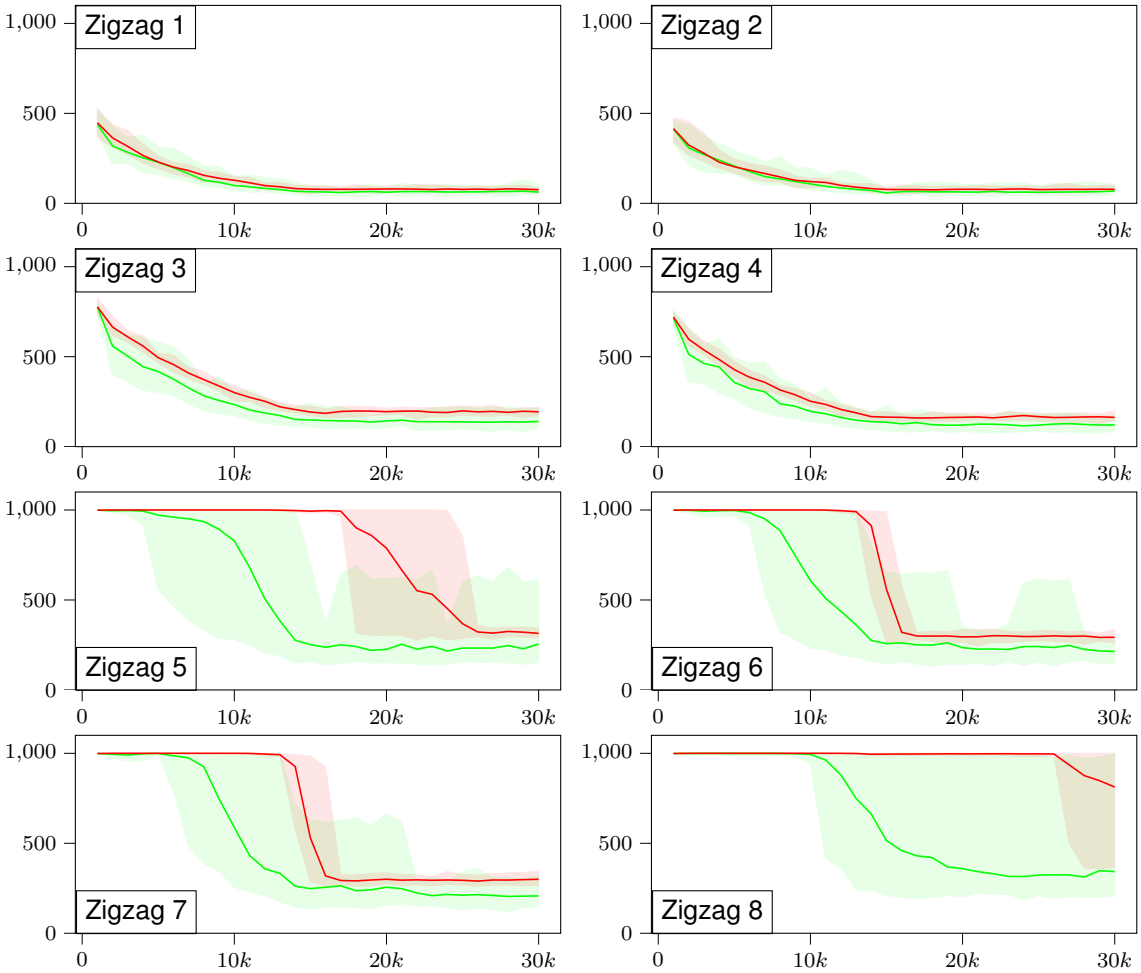


Figure 8.6: The number of safety violations throughout RL in the *zigzag* gridworlds. The y-axes display the number of violations and the x-axes display the episodes at which the policies are evaluated. The green plots represent the number of violations under the shielded policies, whereas the red plots represent the same under unshielded policies.

atively long paths to be traversed and the gained rewards are sparse. Hence, learned safety shields may benefit RL in environments with sparse rewards, where safety violations may prevent the agents from visiting states that give a positive reward. The decreases in the number of safety violations, as shown in Figure 8.6, match the observations on the performance increases illustrated in Figure 8.5.

On the negative side, note that the growth of the return is steeper for unshielded RL. For example, considering the environment *Zigzag 4*, it takes about 10000 episodes to reach an average return greater than 0 in the shielded case and it takes 15000 episodes in the unshielded case. Hence, there are 3000 episodes between first reaching the goal and reaching it more consistently for shielded RL, whereas unshielded RL only requires 1000 episodes to make this jump in performance.

Furthermore, consider the range between the minimum and the maximum return that is depicted by the shaded areas in the figures. The minimum return of unshielded RL is often lower than the minimum return of shielded RL even though it performs better on average. There is more variance in the return obtained by shielded RL. Also, the range between the minimum and the maximum number of safety violations is high for *Zigzag 8* until the end of training. This could result from learning of MDPs that do not sufficiently capture safety-relevant information.

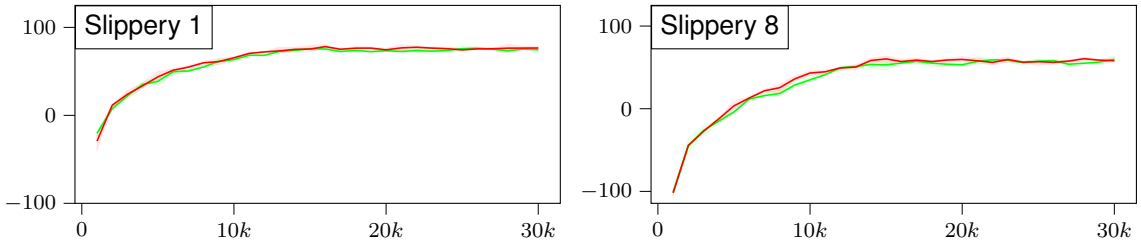


Figure 8.7: Return gained by intermediate policies throughout RL in the *slippery shortcuts* gridworlds. The y-axes display the return and the x-axes display the episodes at which policies are evaluated. Green plots: shielded agent, red plots: unshielded agent.

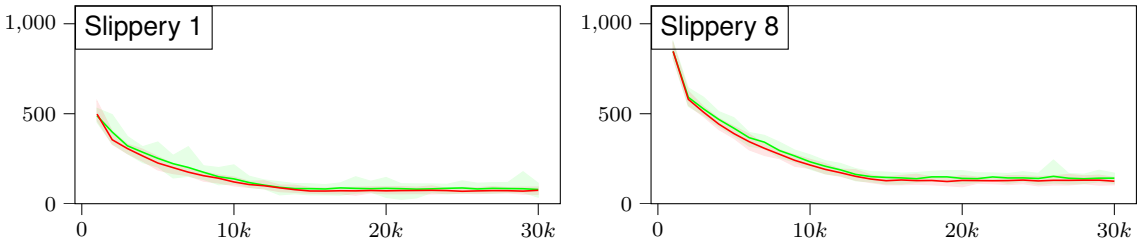


Figure 8.8: The number of safety violations throughout RL in the *slippery shortcuts* gridworlds. The y-axes display the number of violations and the x-axes display the episodes at which the policies are evaluated. Green plots represent the shielded agent, while red plots represent the unshielded agent.

8.3.5 Slippery Shortcuts Gridworlds

Next, we examine the performance of shielded and unshielded RL in the *slippery shortcuts* gridworlds illustrated in Figure 8.3. Figure 8.7 shows the average returns, and Figure 8.8 shows the number of safety violations gained by RL agents throughout learning. In contrast to the *zigzag* gridworlds, there is hardly any difference between the shielded and the unshielded configurations for any size of the environment, neither for the return nor for the number of safety violations. Therefore, we only printed the instances of *slippery 1* and *slippery 8*. Moreover, there is very little variability, as the minimum and maximum returns (safety violations) are very close to their average. Hence, performance is mostly governed by RL and shielding has little influence. In these environments, the agents succeed in finding safe paths without requiring assistance from a shield. This may result from the pits being farther away from optimal paths, as compared to the *zigzag* gridworlds.

8.3.6 Wall Gridworlds

In the following, we discuss the experiments performed on the *walls* gridworlds of Figure 8.4. Figure 8.9 shows the returns, and Figure 8.10 shows the number of safety violations gained at different stages of learning. As for the *zigzag* gridworlds, we see hardly any difference between shielded and unshielded RL for the three smallest environments, whereas shielded RL performs better in the larger environments. Unlike before, however, there is less variability in the performance and number of safety violations of shielded RL. In Figure 8.4, we can see that the slippery tiles are farther away from the optimal route, which is also true for the larger *walls* gridworlds. As a result, learned MDPs do not need to be as accurate with respect to probability estimations for effective shields to be created. Especially for the largest example, *Wall 8*, shielding improves performance and reduces the number of safety violations considerably. It takes about 18000 episodes to learn a policy that consistently reaches the goal in all 30 repetitions of the corresponding experiments. In contrast, unshielded RL fails to consistently find a good policy even after 30000 episodes.

Finally, let us investigate a potential reason for performance improvements resulting from shielding or the absence thereof. In Figure 8.11, we show the size of learned MDPs compared to the size of the

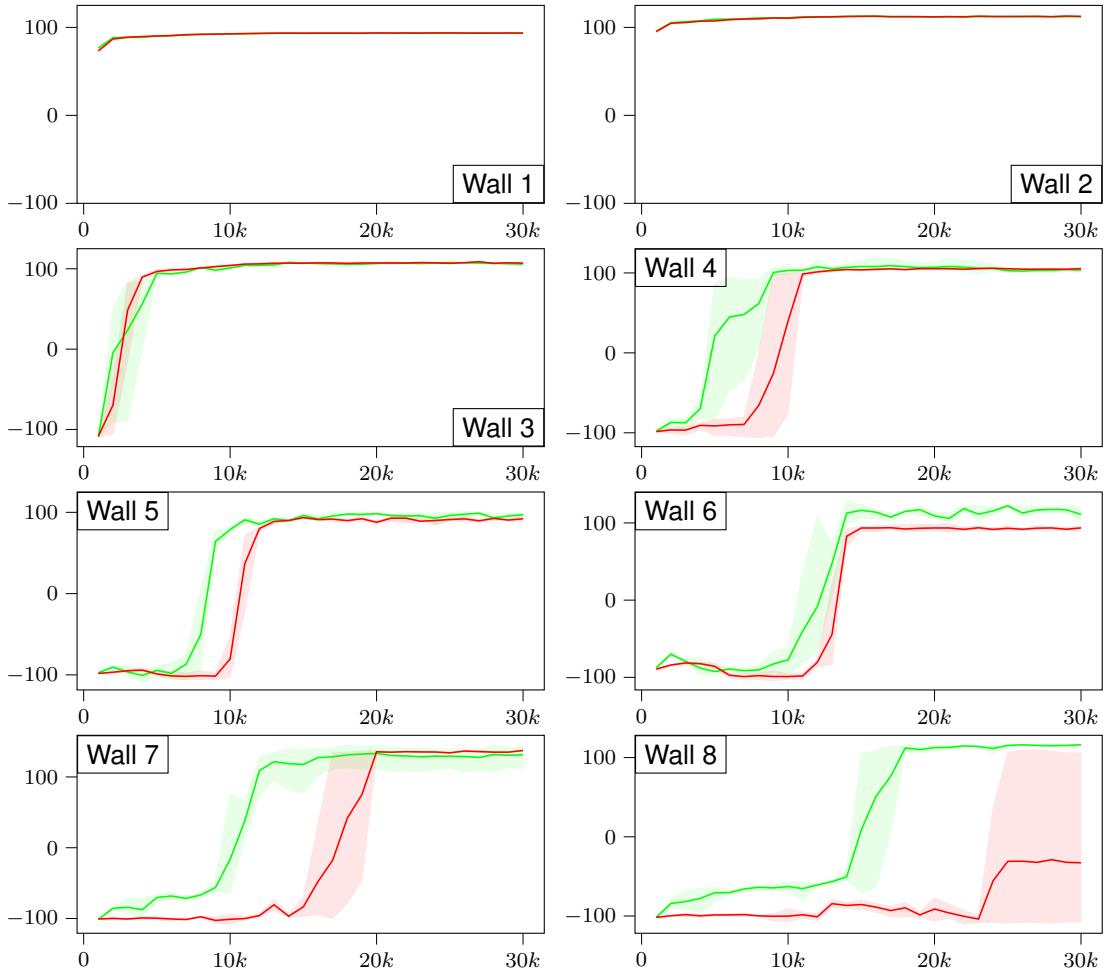


Figure 8.9: Return gained by intermediate policies throughout RL in the *walls* gridworlds. The x-axes display the return and the y-axes display the episodes at which policies are evaluated. Green plots: shielded agent, red plots: unshielded agent.

walls environments, i.e., the number of tiles in every environment. Since the environment size is constant in an experiment, it is shown as a black straight line. The learned MDP size, measured in the number of states, generally increases throughout the learning due to more information getting available. It can be seen that for the first three environment sizes, the final learned MDPs are slightly larger than the environment. Hence, these MDPs cannot represent the environments and their safety-relevant features more efficiently than a Q-table. The fact that learned MDPs are even larger than the environments from which they are learned results from two properties of our learning setup. First, MDPs learned by IOALERGIA only converge in the large sample limit to the true underlying MDPs [144]. There are no guarantees for MDPs learned from finite amounts of data. Second and more importantly, abstraction introduces non-determinism, while MDP learning basically performs a determinization of the resulting non-deterministic MDP. This determinization causes the number of states to increase, similar to the construction of belief MDPs from partially observable MDPs [43] which we explored in Chapter 7.

When the environment is larger than the learned MDP modeling safety-relevant features of the environment, shielding improves RL performance. This holds for all environments from *Wall 4* throughout *Wall 8*. Comparing Figure 8.9 or Figure 8.10 with Figure 8.11, we can see that the larger the size difference between the environment and the learned MDP is, the larger the performance impact or reduction in the number of safety violations.

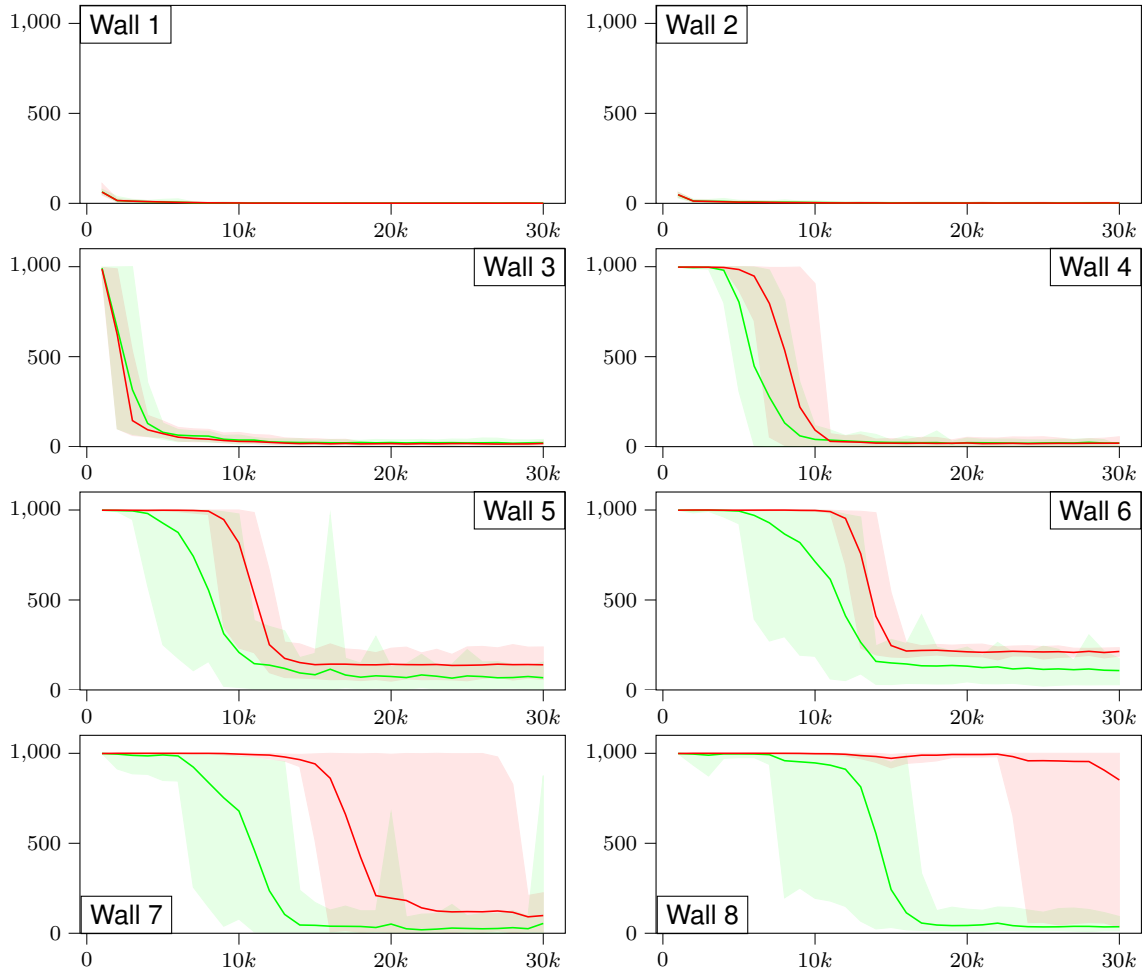


Figure 8.10: The number of safety violations throughout RL in the *walls* gridworlds. The y-axes display the number of violations and the x-axes display the episodes at which the policies are evaluated. Green plots: shielded agent, red plots: unshielded agent.

Discussion. We conclude this section with a discussion of the main results of our experiments and some insights that we gained. Our results show that learned shields can improve RL performance as illustrated by our first and third set of experiments. In the case of the *zigzag* gridworlds, the agent has to traverse along tiles located closely to pits in order to reach the goal. Therefore, a shield is able to prevent many safety violations. In the case of the *walls* gridworlds, we observed that learning shields especially pays off when the learned safety MDP is much smaller than the complete environment.

Scalability. We postulate that the proposed approach can be used with deep-RL methods in more complex environments, such as computer games [155], given the existence of an abstraction function that is able to encode safety-relevant information. Such a mapping would only encode information that is relevant to the safety properties and would not be affected by the high-dimensional observations, as it would operate on a symbolic domain that is suitable for automata learning.

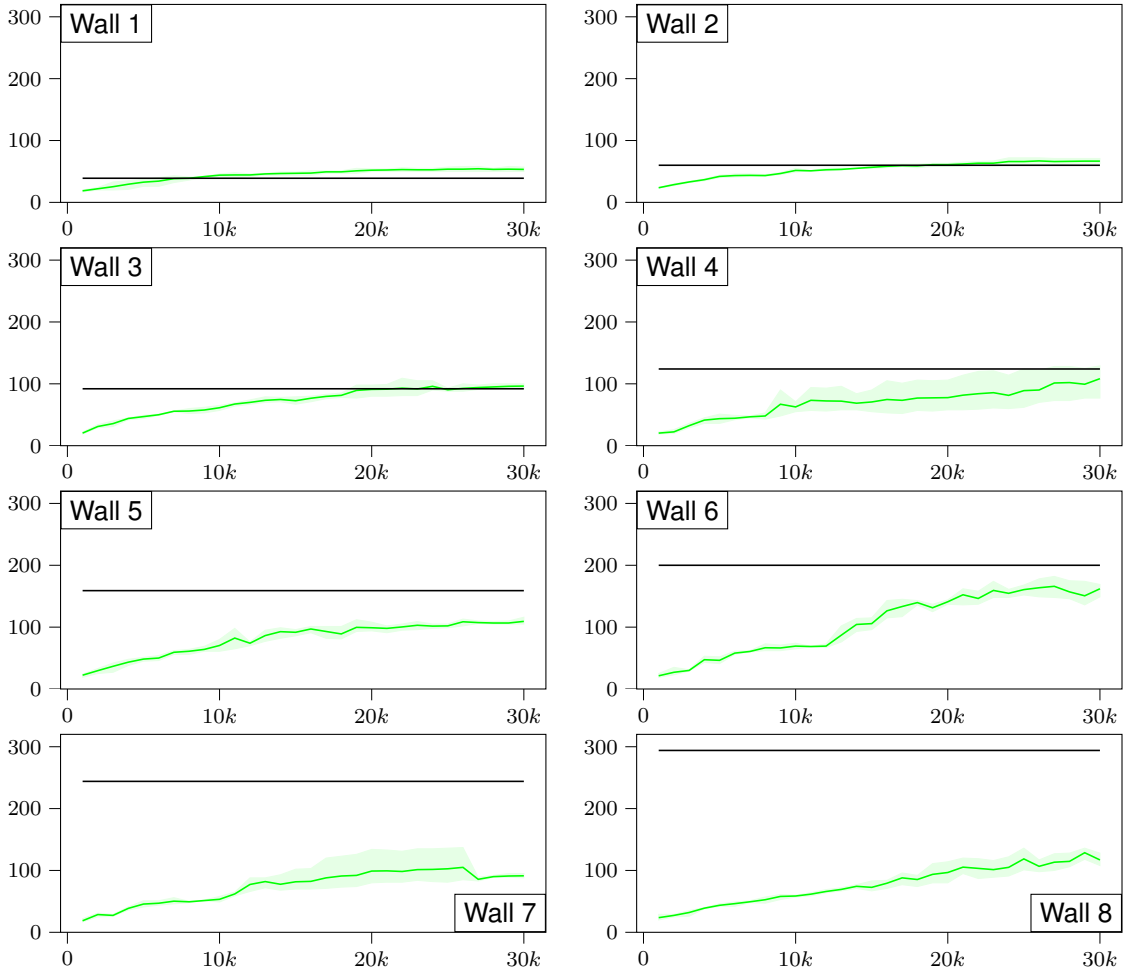


Figure 8.11: Average size of learned MDPs for the *walls* gridworlds (y-axes) plotted in green compared to the size of the environment plotted as a black line. The x-axes display the episode at which the MDP size was measured.

8.4 Concluding Remarks

The existence of an environment model is a prerequisite for shielding, however, such models might not always be available. To combat this issue, in this chapter we have presented an approach for iterative safe reinforcement learning via learned shields. During the training of an RL agent, we learn abstract models that capture safety-relevant dynamics of the environment and continuously update shields that prevent safety violations during execution. We showed that shielding with learned models can result in fewer safety violations throughout the training and even lead to higher overall rewards.

RQ 3.1 Can automata learning help reinforcement learning?

We showed how automata learning can be used to automatically learn abstract models of the environment that encode the safety-relevant information. Those learned models are then used to compute shields that augment the RL agents' behavior during training, resulting in fewer safety violations. Learned models only capture the safety-relevant aspects of the environment, which improves the scalability of model learning and makes the proposed approach applicable to more complex environments, assuming the existence of a safety-relevant abstraction function.

9

Modelling and Testing of Deep-RL Agents in Continuous Stochastic Environments

Declaration of Sources

This chapter is based on our work "Learning Environment Models with Continuous Stochastic Dynamics – with an Application to Deep RL Testing" which was presented at ICST2024 [242].

9.1 Motivation

In previous chapters, we explored various applications of automata learning in the machine learning domain. We used automata learning to model the input-output behavior and analyze RNNs in Chapter 5 and in Chapter 6, and we have shown how automata learning can be combined with reinforcement learning in discrete environments in Chapter 7. However, machine learning algorithms are usually used to solve problems in high-dimensional domains, where analytical solutions require immense human effort.

Model-free deep reinforcement learning (DRL) has proven successful in solving complex sequential decision-making problems in high-dimensional, probabilistic environments. However, the decision-making of DRL systems is highly opaque and the lack of explainable models limits their acceptance in promising application areas like autonomous mobility, medicine, or finance. Difficulties in testing neural-network-based policies further hamper the widespread adoption of DRL.

Thus, it is important to have methods and tools available that automatically learn environmental models under the control of an agent. However, the high dimensionality of the observed environmental states and the environment's complex dynamics render a direct application of automata learning infeasible. To that end, we developed **CASTLE - Clustering-based Activated pasSive automaTa LEarning**, a novel approach for learning environmental models, which is designed for environments with complex dynamics and continuous state spaces.

9.1.1 Guiding Idea

CASTLE combines abstraction mapping, passive automata learning, and active sampling of new trajectories to learn accurate models of high-dimensional environments. More precisely, we use dimensionality reduction and clustering to compute mappers that map from the unbounded observation set to a discrete

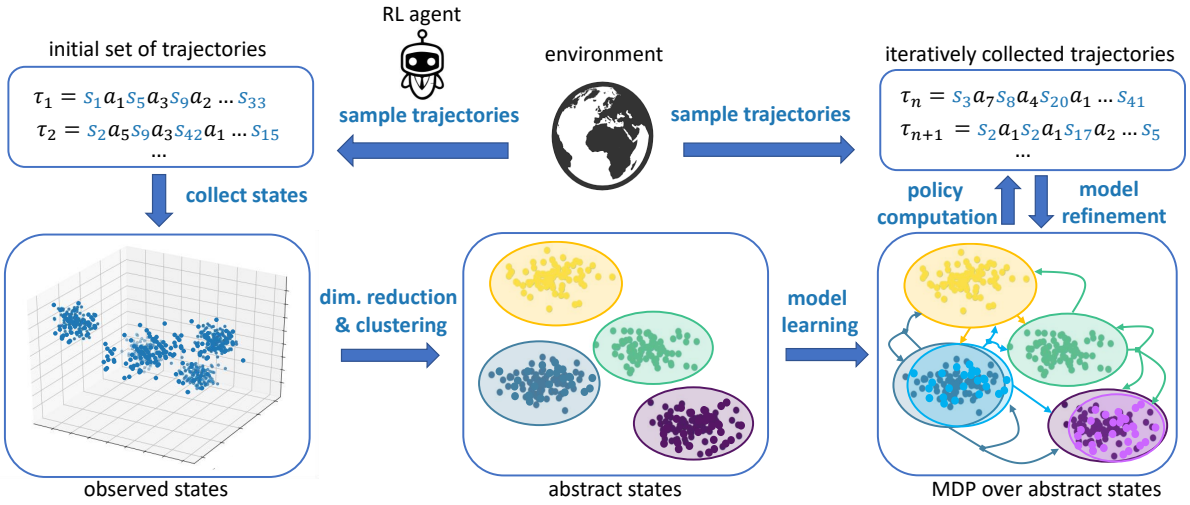


Figure 9.1: Overview of the algorithm CASTLE for learning MDPs modeling environments with continuous stochastic dynamics

set of values and use passive automata learning over these abstracted observations to learn models of the environment. Since these models are learned under strong abstraction that comes with an inherent information loss, we actively sample the environment with our learned model and iteratively improve the learned model. This process is abstractly visualized in Figure 9.1. The left-hand side of Figure 9.1 depicts the initial model learning, where CASTLE is given a multiset of trajectories sampled by an existing, potentially not optimal RL agent. Those trajectories are then abstracted with computed dimensionality reduction and clustering functions. The abstracted trajectories are used to learn the initial abstract model, which is then continuously refined, as abstractly visualized in the right-hand side of Figure 9.1.

Since learned abstract models should accurately capture the dynamics of the underlying continuous stochastic environment, we evaluate them through probabilistic model checking, that is, by deriving policies from learned MDPs that solve control tasks from OpenAI gym [33]. To further demonstrate the generalization capabilities of CASTLE, we present a coverage-based testing strategy based on the learned models, in which the learned models are used to guide deep-RL agents under test to identified areas of interest.

Structure. This chapter is structured as follows: In Section 9.2 we present the method behind CASTLE. We show how CASTLE can be used for differential testing in Section 9.3. We then evaluate CASTLE on RL tasks in Section 9.4 and use it for differential testing in Section 9.5. Finally, we present concluding remarks in Section 9.6.

9.2 Method

Preliminaries. We rely on the standard definitions of MDPs and POMDPS outlined in Sections 2.1.2 and Section 7.2, as well as the IOALERGIA passive automata learning algorithm outlined in Section 3.5.

Setting. We consider settings where an agent performs an episodic task in an environment modeled as an MDP $\mathcal{M} = \langle \mathcal{Q}, I, O, \delta, \lambda, \mathcal{Q}_0 \rangle$. An episodic task is a task that has an end, i,e., it ends in a terminal state. An episode is a sequence of agent-environment interactions from a randomly distributed initial state $q_0 \in \mathcal{Q}_0$ of $\mathcal{M}$ to a terminal state. An episode completes the task to be learned if it ends in a (terminal) goal state. If an episode ends in a bad (terminal) state, it fails to complete the task. In our setting, $\mathcal{M}$ is an MDP with continuous state space $\mathcal{Q} \subseteq \mathbb{R}^h$ and unknown stochastic dynamics and structure. Thus, we do not assume knowledge about the reachable states or transitions of $\mathcal{M}$.

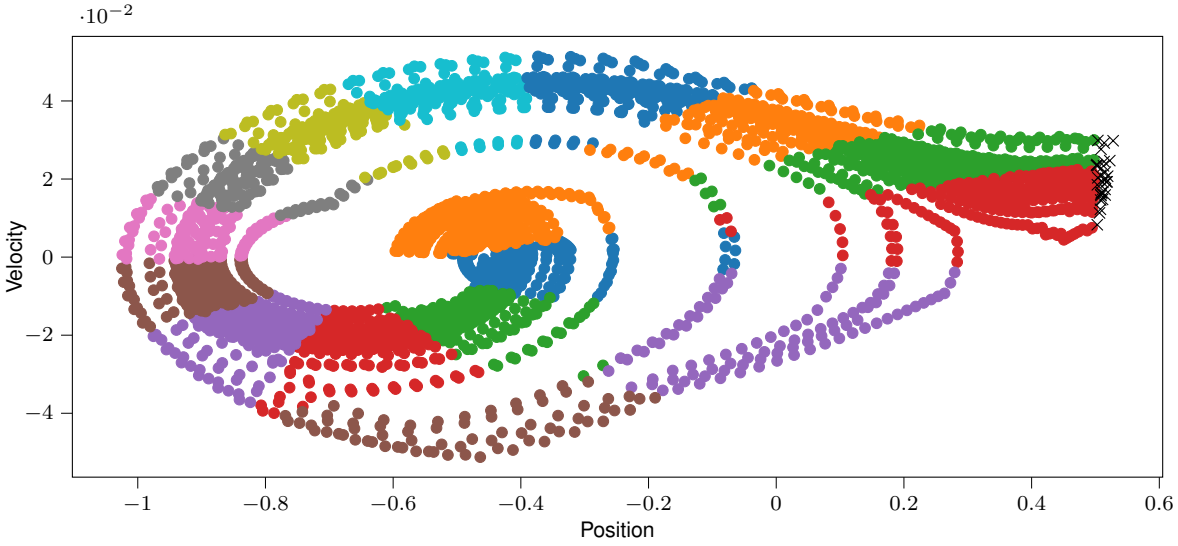


Figure 9.2: States observed in the Mountain Car environment. Colors represent clusters and x-markers indicate goal states.

We are given a multiset of trajectories $\mathcal{T}$, where $\tau = s_0 a_1 s_1 \cdots a_{n-1} s_{n-1} a_n s_n \in s_0 \times (I \times \mathcal{Q})^*$, $s0 \in \mathcal{Q}_0$, sampled from $\mathcal{M}$ when executing a potentially non-optimal policy on $\mathcal{M}$. We assume that a subset of $\mathcal{T}$ are successful trajectories that end in a goal state. The trajectories $\mathcal{T}$ serve as a starting point for passive automata learning.

Example. To illustrate the setting, consider the well-known Mountain Car environment available in OpenAI Gym [33], where the goal is to push a car up a hill. In this environment, the RL agent needs to move the cart back and forth to acquire sufficient momentum to reach the top of the hill. The state space of $\mathcal{M}$ consists of two real-valued variables representing the x-position of the car and its velocity. The agent has the following three actions to interact with the environment: accelerate to the left, accelerate to the right, and do nothing. In the initialization of an episode, the x-position of the car is randomly sampled from the interval $[-0.6, -0.4]$. An episode is successful if the car reaches position 0.5 (defines the goal terminal states in $\mathcal{M}$). After 200 time steps (defines the bad terminal states), the episode terminates as unsuccessful. Figure 9.2 displays the states (current velocity and position) observed during the execution of a policy π over 30 episodes. Black x-markers indicate goal states at the x-position 0.5. Colors in Figure 9.2 represent different clusters identified over the observation space.

Problem statement. Consider the setting as discussed above. The goal of our approach is to learn a concise model $\mathcal{M}_a = \langle Q_a, I, O_a, \delta, \lambda_a, q_{0a} \rangle$ in the form of an abstract deterministic labeled MDP that models the environment $\mathcal{M}$. The model $\mathcal{M}_a$ should be so accurate that a policy $\pi_a : Q_a \to I$ that solves the task to be learned in $\mathcal{M}_a$, also successfully solves the task when being executed on $\mathcal{M}$.

Overview of the learning process. Our approach consists of two phases: an initialization and a fine-tuning phase. The initialization phase prepares the trajectories in the continuous state space so that they can be used to learn an abstract, concise MDP model $\mathcal{M}_a$. Fine-tuning computes a policy based on $\mathcal{M}_a$ and uses it to collect additional trajectories to fine-tune $\mathcal{M}_a$ with more information about the environment $\mathcal{M}$. The policy $\pi_a : Q_a \to I$ for $\mathcal{M}_a$ is automatically computed by solving a reachability objective through probabilistic model checking. The computed policy π_a selects actions that have the maximal probability of successfully completing the task. Note that the subscript a stands for *abstract* and serves to distinguish the learned abstract MDP $\mathcal{M}_a$ over abstract states Q_a from the concrete MDP $\mathcal{M}$ over real-valued states $\mathcal{Q}$.

9.2.1 Initial Model Learning

The initialization phase of CASTLE sets up automata learning and learns a first MDP. For the remainder of this section, let $\mathcal{M} = \langle \mathcal{Q}, I, O, \delta, \lambda, \mathcal{Q}_0 \rangle$ be the MDP underlying the environment with state space $\mathcal{Q} \subseteq \mathbb{R}^h$, where h is the size of the state vectors, and let $\mathcal{T}$ be a multiset of trajectories $(\mathcal{Q} \times I)^* \times \mathcal{Q}$. Our goal is to learn an abstract deterministic labeled MDP $\mathcal{M}_a$ serving as an initial model of $\mathcal{M}$ from the trajectories $\mathcal{T}$.

First, our approach transforms the trajectories in $\mathcal{T}$, which are sequences of real-valued states and actions, into observation traces, which are sequences of abstract observations and actions. The transformation consists of the steps: (1) dimensionality reduction and scaling, (2) clustering, and (3) additional labeling of states. After performing these data-processing steps, we compute an initial model $\mathcal{M}_a = \langle Q, I, O, \delta, \lambda, q_0 \rangle$ using IOALERGIA as a final fourth step. In the following, we discuss the individual steps in detail.

1. Dimensionality Reduction and Scaling. For high-dimensional state spaces, we apply a dimensionality reduction $sd : Q \to Q^d$ to transform $Q \subseteq \mathbb{R}^h$ to $Q^d \subseteq \mathbb{R}^d$ with $h > d$. We denote with Q_T and Q_T^d the set of all states and reduced states contained in $\mathcal{T}$. While any standard technique like principle component analysis (PCA) [139] can be applied, we propose to apply a dimensionality reduction approach that works with the given trajectories in $\mathcal{T}$ instead of working only with the observed states. Thus, we propose to guide the dimensionality reduction with the actions taken in the observed states. For control applications, the information loss incurred by dimensionality reduction is low, if the action taken in a state can be predicted with the same accuracy in $\mathbb{R}^d$ as in $\mathbb{R}^h$.

Linear discriminant analysis (LDA) [28] is a natural fit for classifying states according to actions. LDA can compute discriminant functions mapping states to actions such that they match the state-action pairs of the demonstration trajectories. That is, we apply LDA to learn a classifier from states to actions using the demonstrations $\mathcal{T}$. This enables reducing the dimensionality by projection to the d most discriminative axes.

Alternatively, we propose a semi-manual technique for dimensionality reduction using decision trees (DT) [28]. If domain knowledge is available, it can be used to extract d features from the states and train DTs with a bounded size to classify states to actions. If the classification accuracy of DTs trained in $\mathbb{R}^d$ (reduced states) is similar to the accuracy in $\mathbb{R}^h$ (original states), we assume that the d extracted features are sufficient for the task at hand. The choice of the d features allows a dimensionality reduction with a small loss of information.

Note that dimensionality reduction is optional and can be skipped if the number of dimensions is low. After dimensionality reduction, the reduced states can be further prepared for ideal clustering by applying power transformation [280] and scaling the state data to zero mean and unit variance.

2. Clustering. In the next step, CASTLE applies a clustering function $clust : \mathcal{Q}^d \to K$ to assign cluster labels $K = \{1, \ldots, k\}$ to dimensionality-reduced states. To facilitate an application in CASTLE, the clustering approach should enable estimating cluster membership via distances or probabilities, and it should be efficient. CASTLE exploits such distances to simulate learned MDPs during sampling; see Section 9.2.2. The popular k-means algorithm [135] satisfies these requirements.

3. Labeling. CASTLE assigns a set of labels to the states in a trajectory in reduced dimensions. The labels are observations that are crucial for computing a policy over a learned MDP.

We define a labeling $lab : \mathcal{Q}^d \times \mathbb{N}_0 \to 2^O$, for a set of observations $O = \{init, goal, bad\} \cup K$. This function assigns labels to (s, i) where s is a state at index i in a trajectory. A pair (s, i) is labeled with $init$, if $i = 0$, i.e., s is an initial state in a trajectory. For any other (s, i), the cluster label $clust(s)$ is contained in $lab(s, i)$. Furthermore, we have $goal \in lab(s, i)$ if s is a goal state or $bad \in lab(s, i)$ if s is a bad state. Both depend on i, since tasks may define a time limit for successful completion. If

additional domain knowledge is available, it can be used to assign additional labels to states, e.g., to indicate potentially dangerous situations.

4. MDP Learning. To learn an MDP from trajectories $\mathcal{T}$, CASTLE transforms them into observation traces $\mathcal{T}_O$, by sequentially applying dimensionality reduction, scaling, clustering, and labeling. Given $\mathcal{T}_O$ as input, IOALERGIA computes a deterministic labeled MDP $\mathcal{M}_a$, providing an abstract representation of the environment dynamics. Introducing the $init$ label for initial states enables modeling environments where the initial states are randomly distributed. With that, we ignore the concrete initial environment states, such that the transitions from the learned initial state (labeled $init$) include the distribution of initial environment states.

Example. The colors in Figure 9.2 indicate clusters of states derived with k-means and *k=16* and the black x-markers indicate goal states. Thus, the states on the right of the figure have two labels: a cluster label and the label $goal$. All other states have a single label corresponding to a cluster unless they are reached at the end of an episode, in which case they are labeled bad to indicate a timeout. Thus, states observed at different times in a trajectory may get labeled differently. An example of an observation trace is $\{init\} \cdot left \cdot \{c_1\} \cdot right \cdot \{c_1\} \cdots right \cdot \{c_{15}, goal\}$. It starts with $init$ and then alternates between actions and observations that include cluster labels. The final observation includes $goal$ to indicate a successful episode. With this information, IOALERGIA learns temporal dependencies between observations and actions. In this example, IOALERGIA usually learns models of varying sizes, ranging from 28 to 52 states, depending on the provided traces and computed abstraction.

9.2.2 Model Fine-Tuning

The fine-tuning phase of CASTLE incrementally improves the learned labeled MDP $\mathcal{M}_a$ that models the environment $\mathcal{M}$. In a nutshell, this phase iteratively (1) computes a policy that can solve the task in $\mathcal{M}_a$ via probabilistic model checking, (2) uses the policy to sample new trajectories, and (3) learns a new, improved model with the extended multiset of trajectories.

The fine-tuning phase is based on the approach proposed by Aichernig and Tappler [4]. In contrast to the original approach, which works solely on abstract observations, our fine-tuning takes the concrete state space and the uncertainties stemming from clustering into account. In the following, we detail the individual steps of this phase.

1. Policy Computation. Given $\mathcal{M}_a = \langle Q_a, I, O_a, \delta, \lambda_a, q_{0a} \rangle$, we aim to compute a (deterministic) policy $\pi_a : Q_a \to I$ that maximizes the probability of completing the task successfully, i.e., reaching a goal state in $\mathcal{M}_a$. Thus, for any state $s \in Q_a$, the policy π_a picks the action $a_\pi = \pi_a(s)$ that maximizes this probability. Such policies can be automatically computed via probabilistic model checking (using tools like PRISM [128]). Let $p_{s,a} = P_{\max}(\mathbf{F}\ goal | s, a)$ be the maximal probability of reaching a goal state from state s when executing action a, where $\mathbf{F}$ denotes the *eventually* operator. For any state $s \in Q_a$, we have $a_\pi = \pi(s)$ with $a_\pi = \max_{a \in I} p_{s,a}$.

2. Sampling. In the next step of the iterative model refinement phase, CASTLE uses the policy $\pi_a : Q_a \to I$ over the current $\mathcal{M}_a$ to sample additional trajectories in $\mathcal{M}$. The newly sampled trajectories are added to the existing trajectories $\mathcal{T}$ and are used in the next step to improve the accuracy of $\mathcal{M}_a$.

During sampling, the learned MDP $\mathcal{M}_a = \langle Q_a, I, O_a, \delta, \lambda_a, q_{0a} \rangle$ is simulated in parallel with the environment $\mathcal{M} = \langle \mathcal{Q}, I, O, \delta, \lambda, \mathcal{Q}_0 \rangle$ with actions selected via the policy $\pi_a : Q_a \to I$.

During sampling, the MDP $\mathcal{M}_a$ is treated similarly to a partially observable MDP. To account for inaccuracies in $\mathcal{M}_a$, we adopt the notion of belief states. A belief state B is a distribution over the states Q_a, i.e., at any time step, $\mathcal{M}_a$ is in state $s_a \in Q_a$ with probability $B(s_a)$. The belief state update is based on the structure of learned MDP $\mathcal{M}_a$ and the environment state reached after a step. At time step i, after having taken an action act and the environment having moved to a state $s' \in Q$ in cluster k', we

Algorithm 9.1 Sampling with a policy computed from a learned MDP

Input: $\mathcal{M} = \langle \mathcal{Q}, I, O, \delta, \lambda, \mathcal{Q}_0 \rangle$, model $\mathcal{M}_a = \langle Q_a, s_{a0}, I, \delta_a, \lambda_a \rangle$, policy $\pi : Q_a \to I$, clusters K, max. belief size b_n

Output: A sampled trajectory τ

1: $s \in \mathcal{Q}_0 \leftarrow$ `reset()`
2: $\tau \leftarrow s$, $B \leftarrow \{s_{a0} \mapsto 1\}$
3: **while** s is not terminal **do**
4: $\quad ba \leftarrow \{act \mapsto \sum_{s_a \in Q_a, \pi_a(s_a)=act} B(s_a) | act \in I\}$
5: $\quad$ sample $act \sim ba$
6: $\quad s \leftarrow \texttt{step}(act)$
7: $\quad \tau \leftarrow \tau \cdot act \cdot s$
8: $\quad dists \leftarrow \{k \mapsto dist(sd(s), centroid(k)) | k \in K\}$
9: $\quad cDistr \leftarrow$ *cdf. of* $\mathcal{N}(mean(dists); stddev(dists)^2)$
10: $\quad B' \leftarrow \{s_a \mapsto 0 | s_a \in Q_a\}$
11: $\quad$ **for** $s_a \in supp(B)$, $s'_a \in supp(\delta_a(s_a, act))$ **do**
12: $\quad\quad k' \leftarrow \lambda_a(s'_a) \cap K$
13: $\quad\quad B'(s'_a) \leftarrow B'(s'_a) + B(s_a) \cdot (1 - cDistr(dists(k')))$
14: $\quad B' \leftarrow \{s_a \mapsto p \in B' | p$ *is in the* b_n *largest values of* $B'(\cdot)\}$
15: $\quad B' \leftarrow \{s_a \mapsto \frac{p}{\sum_{s'_a} B(s'_a)} | s_a \mapsto p \in B'\}$
16:
17: **return** τ

update the belief state B to B' to include states s'_a with $\delta_a(s_a, act, s'_a) > 0$ for $s_a \in supp(B)$. That is, we move to states reachable in $\mathcal{M}_a$. The probabilities $B'(s'_a)$ are the product of $B(s_a)$, i.e., the previous state probability, and a term that is inversely proportional to the distance between the cluster centroid of k' (the reached cluster) and the centroid of the cluster in $\lambda_a(s'_a)$.

The following scenario highlights why keeping track of a single state $s_a \in Q_a$ is not sufficient. Suppose that the environment $\mathcal{M}$ is in a state $s \in \mathcal{Q}$ and the learned model $\mathcal{M}_a$ is in a state $s_a \in Q_a$. Next, the environment moves from state s to s' via action act. The corresponding cluster labels of the states s and s' are k and k', respectively. Ideally, in the learned model $\mathcal{M}_a$, there is a unique s'_a corresponding to s' identified by $\delta_a(s_a, a, s'_a) > 0$ with $k' \in \lambda_a(s'_a)$. However, since the learned models are not perfectly accurate, especially during early iterations, this is not always the case.

Algorithm for sampling of trajectories. Algorithm 9.1 formalizes our approach to sample a trajectory from $\mathcal{M}$ using a policy computed from a learned model $\mathcal{M}_a$. We follow the OpenAI Gym [33] conventions and use the operations `reset` and `step` to change the current state s of the environment $\mathcal{M}$. The function `reset` resets $\mathcal{M}$ to an initial state and returns this state. The function `step` takes an action a as input, executes a, and returns the reached state s'.

Algorithm 9.1 first resets the environment, adds the initial state to the sampled trajectory τ, and initializes the belief state B to include only the initial state of $\mathcal{M}_a$ (lines 1 and 2).

We then sample experiences from $\mathcal{M}$ until reaching a terminal state (Line 3). Line 4 transforms the belief state into a distribution over actions by mapping states to actions chosen by the policy π_a. The next three lines sample an action, execute it in the environment, and append the new state-action pair to the trajectory τ. Hence, the combination of the current belief state and a deterministic policy π_a yields a probabilistic policy.

Starting in Line 8, we begin the update of the belief state. First, we compute the Euclidean distances from all cluster centroids to the current environment state in reduced dimensions. We then fit a normal distribution over distances, which we empirically found to be a good fit. After that, we initialize the next belief state B' and iterate over all states reachable in $\mathcal{M}_a$ (Line 11). In Line 13, we add the contribution of s'_a to the next belief state B' as the product of the previous belief state $B(s_a)$ and the inverse of

the distance probability of the cluster k' labeling s'_a. We use the inverse to favor short distances and we ignore $\delta_a(s_a, act)$ so that we only consider the actual environment information for the belief state update. In Line 14 we discard states with low probability. Finally, Line 15 normalizes the belief states to a distribution.

3. Learning and Stopping. In each iteration of the fine-tuning phase, we sample n_{samples} additional trajectories from $\mathcal{M}$, as outlined above. To learn a more accurate model $\mathcal{M}_a$ from the additional information, CASTLE takes the newly collected trajectories $\mathcal{T}^{new}$ and transforms them into observation traces $\mathcal{T}_O^{new}$. This is done by sequentially applying dimensionality reduction, scaling, clustering, and labeling as outlined in Sect. 9.2.1. CASTLE adds the new observation traces to the existing multiset of traces $\mathcal{T}_O$ and learns a labeled MDP $\mathcal{M}_a$ with IOALERGIA. After computing a new learned model $\mathcal{M}_a$, CASTLE returns to the policy computation step.

As stopping criteria for the iterations, one can stop either after a fixed number of iterations or upon reaching a goal state (with the computed policy) a specified number of times in the current iteration.

9.3 Differential Testing

This section shows how to apply CASTLE for differential functional testing of RL agents. Note that in our context the performed differential testing differs from the standard definition, in which the same inputs would be provided to different implementations and differences in outputs would be analyzed. More precisely, we compare two RL policies with respect to the number of functional failures they induce and we use cluster coverage for test selection. Differential testing helps us tackle the problem that we have no explicit test oracle. This problem is tied to the issue that it may generally be difficult for an RL policy to solve the task under consideration from a cluster that we cover by testing. In other words, starting from certain clusters, failures may be inescapable. To mitigate this issue, we compare the relative frequencies of failures, thus providing information on which RL policy under test is safer.

Overview. The inputs for our testing methodology are an MDP learned with CASTLE, the generated dimensionality reduction and clustering, and two DRL agents under test. Given these inputs, we first identify *important clusters*. For every important cluster c, we run a low number of CASTLE iterations with c as the target to compute a policy π_c to reach c. After that, we start the actual testing loop, where in every iteration: we use π_c until we reach c, then we let one agent under test take over, by executing its policy and checking if it induces a functional failure. We call this an individual **test run** that an agent may **pass** or **fail**. When π_c fails to reach c before episode termination, we say that the test run is inconclusive. Between each iteration, we switch the agents to get an equal number of tests for each. The loop terminates after a maximum number of test runs or we find that there is a statistically significant difference in the occurrence of failures between the two agents. Below, we detail the individual steps involved in testing.

To **determine important clusters**, we adopt the notion of state importance [246]. The importance of a state s is $I(s) = \max_a Q(s,a) - \min_a Q(s,a)$, where Q is a Q-function learned by an RL agent. Hence, we assume access to a Q-function-based agent, e.g., trained via DQN [155]. With this state-importance criterion, we identify states in which a certain action must be performed in order to maximize the reward, compared to the states in which the selection of an action might not have much influence on the future reward. In addition to state importance, we consider clusters more important if we observe them more frequently. Thus, we sum the state importance values in every cluster. Given an agent, we execute n_{imp} episodes to gather importance information. Let Q_{imp} be the states observed during these episodes and let K, $clust$, and sd be the clusters, the clustering function, and the dimensionality reduction derived in the initial phase of CASTLE. For a cluster $k \in K$, we define its importance as

$$I(k) = \sum_{s \in Q_{imp} : clust(sd(s)) = k} I(s).$$

Algorithm 9.2 Overview of differential testing with CASTLE

Input: Deep RL policies under test π_0, π_1, target cluster k, clustering $clust$, dimensionality reduction sd, abstract MDP $\mathcal{M}_a$, n_{tref}
Output: Test fail ratio, significant difference information

1: $ag \leftarrow 0$, $fail_0 \leftarrow 0$, $fail_1 \leftarrow 0$, $pass_0 \leftarrow 0$, $pass_1 \leftarrow 0$
2: $\mathcal{M}_a \leftarrow n_{tref}$ iterations of `CASTLE`$(\mathcal{M}_a, clust, sd)$ with k as goal
3: $\pi_{test} \leftarrow$ *policy on* $\mathcal{M}_a$ *to* k
4: **for** $i \leftarrow 1$ **to** max. #test runs **do**
5: $\quad \pi_{agent} \leftarrow \pi_{ag}$
6: $\quad s \in Q \leftarrow$ `reset()`
7: $\quad$ **while** $clust(sd(s)) \neq k$ **do**
8: $\quad\quad s \leftarrow$ control with π_{test}
9: $\quad\quad$ **if** episode is over **then**
10: $\quad\quad\quad$ **continue** with inconclusive verdict
11: $\quad$ **while** episode not over **do**
12: $\quad\quad s \leftarrow$ control with π_{agent}
13: $\quad$ **if** s is a failure state **then**
14: $\quad\quad fail_{ag} \leftarrow fail_{ag} + 1$
15: $\quad$ **else**
16: $\quad\quad pass_{ag} \leftarrow pass_{ag} + 1$
17: $\quad ag \leftarrow ag + 1 \mod 2$
18: $\quad$ **if** `different`$(\frac{fail_0}{fail_0+pass_0}, \frac{fail_1}{fail_1+pass_1})$ **then**
19: $\quad\quad$ **break**

After assigning importance to clusters, we **select** the k_{imp} most important clusters for test execution. We provide pseudo-code for **differential testing with cluster coverage** in Algorithm 9.2. The input for testing is an MDP $\mathcal{M}_a$ learned by CASTLE. In Line 2, we fine-tune $\mathcal{M}_a$ towards reaching the target cluster k by running CASTLE for at most n_{tref} iterations, to make the test policy π_{test} (Line 3) more effective.

Every **test run** first selects one of the agents and resets the environment. Then, we control the agent using the test policy (lines 7 to 10) until reaching cluster k. After that, the RL policy under test takes over the execution until it reaches a terminal state. We propose to decide test run verdicts based on the type of reached terminal state. A policy **passes** a test run if it reaches a goal state, i.e., if it solves the respective task from the covered cluster k. It **fails** a test run if it reaches a bad terminal state. Such a test failure exposes a potential functional fault of the RL policy under test. Testing terminates after a maximum number of test runs or if we detect a significant difference in the ratios of failing test runs between the two agents. For this purpose, we treat the occurrence of test failures as Bernoulli random variables and implement `different` in Line 18 via a two-sided Fisher's exact test [72] with a significance level of 0.05. As test results, we return the ratios of test failures and whether we detect a significant difference. For brevity, we omit explicit **return** statements.

Figure 9.3 shows a graphical representation of test runs in the Lunar Lander environment. We apply CASTLE to derive a policy π_{test} to execute a test run prefix (yellow) that leads to the cluster of interest (shaded area), from where a policy π_{ag} under test tries to land between the flags.

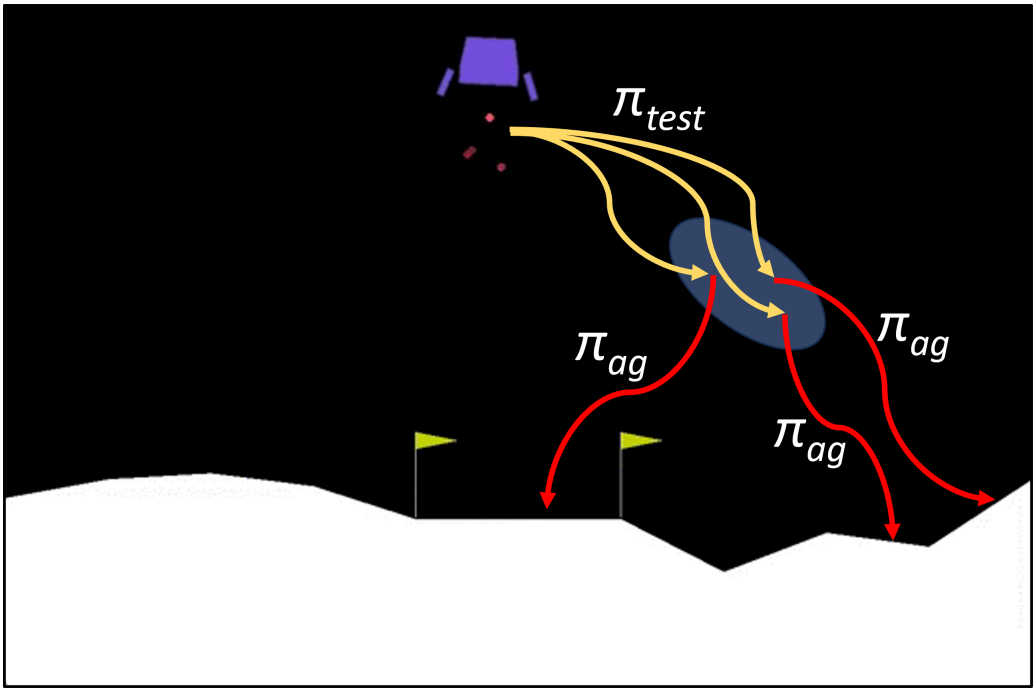


Figure 9.3: Graphical representation of differential testing of lunar lander agents. Yellow trajectories executed with CASTLE reach an interesting cluster (shaded area), while red trajectories are executed with an agent under test.

9.4 Experimental Evaluation of CASTLE

We applied CASTLE to four well-known RL problems [33]: (1) Acrobot, (2) Lunar Lander, (3) Mountain Car, and (4) Cartpole. All reported results are averaged over five experiment runs per environment. Pretrained *stable-baselines3* [210] RL agents that achieved high rewards on their respective tasks were downloaded from HuggingFace[1]. Model learning was done with AALPY's implementation of IOALERGIA [166]. For policy computation over the learned models, we used the model checker PRISM [128]. All experiments were conducted on a laptop with an Intel® Core™i7-11800H CPU at 2.3 GHz with 32 GB of RAM. Source code and instructions to reproduce all experiments are publicly available [159].

Results. Table 9.1 contains parameterized experimental results. The "Demonstration Trajectories" column gives the total number of episodes/trajectories sampled with a DRL agent downloaded from HuggingFace. The column "Demonstration Timesteps" lists the timesteps performed on the environment during the sampling of trajectories, i.e., their combined length. Please note the compact size of the learned models in the column "Final Model Size". In all experiments, each environment state was observed only once (due to the continuous state space). The total number of observed states is thus equal to the number of timesteps. Therefore, if we would directly apply automata learning, the learned MDP would have 2.1×10^5 to 5.6×10^5 states. The learned MDPs of CASTLE have 213 to 1122 states. Next, we discuss the results of the individual experiments in more detail.

Acrobot. Acrobot is a two-link pendulum actuated by a single joint. Its state space consists of 6-dimensional real-valued vectors encoding link angles and angular velocities. The goal is to swing the bottom link of the pendulum up to a target height in as few steps as possible. The task is considered to be solved, if the target can be achieved in 100 steps, shown as a green line in Figure 9.4a. We evaluated our method with two different dimensionality reduction techniques: using a semi-manually created dimensionality reduction via DTs and using LDA. As seen in Table 9.1, both approaches can find a model which allows to solve the task in 42% and 4% of cases. These results are still remarkable given

[1] `https://huggingface.co/`

Table 9.1: Parameterized results for all environments. All values are averages from five experiment runs.

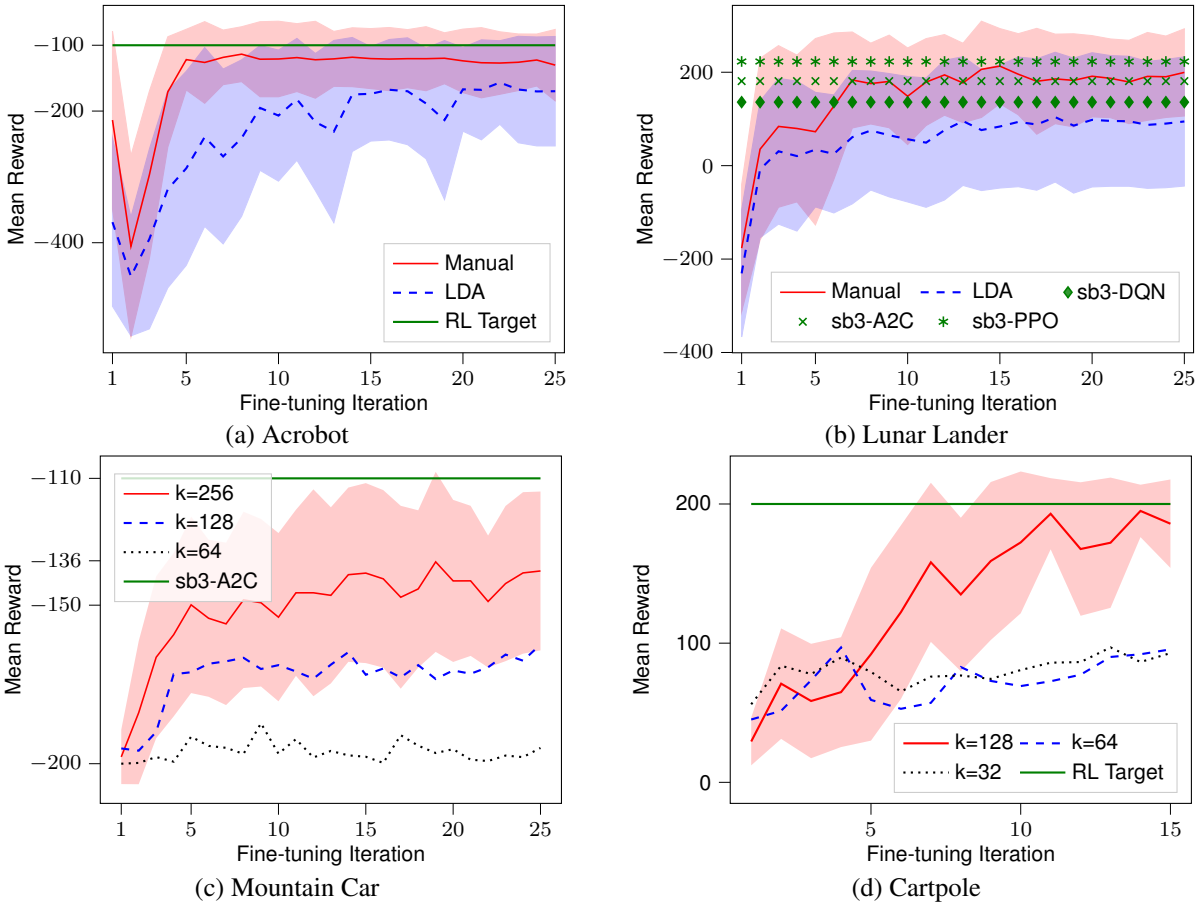
Environment	Demonstration Trajectories	Demonstration Timesteps	Dimensionality Reduction	Clusters	Fine-tuning Iterations	Episodes Per Fine-tuning	Final Model Size	Best Policy Reward	Goal Reached
Acrobot	2500	2.1×10^5	Manual Mapper LDA + Power Transformer	256	25	50	472 482	-113 $\pm$ 41 -156 $\pm$ 63	42% 4%
Lunar Lander	2500	5.6×10^5	Manual Mapper LDA + Power Transformer	1024	25	50	1122 1100	213 $\pm$ 80 103 $\pm$ 138	81% 35%
Mountain Car	2500	3×10^5	Power Transformer	256	25	50	363	-136 $\pm$ 28	42%
Cartpole	2500	5×10^5	Power Transformer	128	15	50	213	195 $\pm$ 18	88%

that our approach computes a control policy to merely maximize the probability of solving the task, but does not necessarily optimize for maximum rewards like RL agents. If we reduce the goal from 100 steps to 130 steps, we observe that our models can compute a policy that reaches this goal in 82% and 35% of cases. Figure 9.4a shows the averaged gained rewards throughout learning. The red line indicates the average reward gained with manual dimensionality reduction and the red-shaded area shows the standard deviation of the gained area. The results for LDA-based dimensionality reduction are shown in blue. We can observe that the models with manual dimensionality reduction yield to a good policy after just 4 iterations of fine-tuning (total of 200 episodes). LDA-based dimensionality reduction leads to policies achieving less stable, but still high rewards.

Lunar Lander. Lunar lander is a classic rocket trajectory optimization problem, whose environment was shown in Figure 9.3. The task is to land the rocket in the landing area as fast as possible. At the beginning of each episode, a random force is applied to the rocket. Its state space consists of 8-dim. vectors. We compared our learned models for two different dimensionality reductions with three DRL agents trained via *stable-baselines3* [202, 205, 208] All *stable-baselines3* agents are able to land the rocket successfully. As shown in Figure 9.4b, the learned models are accurate enough after only 8 fine-tuning iterations to allow the computation of a good policy. The graphs follow the same color coding as for Acrobot, with the exception that the DRL agent results are shown with markers. We observe a performance gap between models computed from the manually-crafted dim. reduction and the LDA-based one. As seen in Table 9.1, the policy constructed with manually-created dim. reduction successfully lands with a probability of 81%, compared to 35% of the LDA-based policy. However, the policies for both models are safe and do not crash the rocket. Through visual inspection, we found that if our policy does not land successfully, it hovers close to the landing position. It is noteworthy that the policy computed with manual dim. reduction even outperforms two of the DRL agents and gets close to the third agent that was trained using PPO [219].

Mountain Car. Mountain car is a control problem, where an agent needs to bring a car to the top of a steep hill in less than 200 steps. Environmental states consist of 2 real values, the car's x-position, and its velocity, therefore we perform no dimensionality reduction. As policies computed from our learned models complete the task quite consistently, we compare to the mean reward of the agent we used to sample demonstration trajectories [203]. This reference is shown as a green line in Figure 9.4c. The dashed and dotted lines depict the mean reward gained when learning a model over $k = 128$ and $k = 64$ clusters, respectively. The red line and red-shaded area show the mean and standard deviation of the reward gained with 256 clusters. In this configuration, our approach with 256 clusters computed an MDP with 363 states that leads to a slightly worse-performing policy than the RL agent (-136$\pm$ 28 compared to -110$\pm$19). Hence, our method successfully learns close-to-optimal policies, even considering that it does not necessarily optimize for rewards, but towards a reachability objective. We further observe that a larger value for k leads to a better policy. This can be attributed to the fact that a larger number of clusters helps to differentiate states during MDP learning.

Cartpole. Cartpole is a classic control problem in which an agent needs to balance a pole attached to a cart. The task is considered solved with a reward of 195 [21], that is, the pole is balanced for at least 195 steps. This translates to a reward of 195 shown as a green line in Figure 9.4d. Its states are encoded as 4-dimensional real-valued vectors. As seen in Table 9.1, our approach solved the game in 15 fine-tuning iterations (750 episodes) with a 213-state MDP, achieving an average reward of 195$\pm$18.



(a) Acrobot (b) Lunar Lander

(c) Mountain Car (d) Cartpole

Figure 9.4: Mean and standard deviation of all experiments over multiple fine-tuning iterations

Like for Mountain Car, we show the mean and standard deviation for one configuration ($k = 128$) in red and for two lower settings of k, we show the mean reward. We again see that a larger k leads to better performance.

Hyperparameter selection. To select the appropriate number of clusters k in k-means, we have applied the elbow method as a starting heuristic. We observed that, for the considered environments, after 128 clusters the k-means inertia decreases. However, after further experimentation, we observed that a higher number of clusters correlates with increased performance of the model but with higher computational costs, especially in the MDP learning. Therefore, we consider the perceived complexity of the environment to select a concrete k. Since we aim for very few assumptions on the environment, we used the number of dimensions of the original state space of each environment as a complexity estimate.

The number of demonstration trajectories was set to 2500 for all experiments. This value was selected to ensure enough data points to compute sufficiently accurate dimensionality reduction and clustering accuracy. The number is further motivated by the number of training timesteps of one of the DRL agents we use for comparison. The Mountain Car A2C agent was trained for 10^6 time steps, which requires at least 5000 episodes, as the maximum episode length is 200. Hence, we decided on using half of that for the initialization phase of our approach. The number of fine-tuning iterations, which we set to 25, and episodes per iteration $n_{samples} = 50$ was selected so that the total number of sampled trajectories is half of the demonstration trajectories. This setting ensures relatively low sampling compared to RL agents, while the results indicate that the fine-tuning of the model can converge to a close-to-optimal solution in a few iterations.

IOALERGIA's ϵ parameter was set to 0.005 for all experiments. Other values showed minimal impact on model quality, which is in line with the observation [41]: "The algorithm behaved robustly with respect to the choice of parameter epsilon". Furthermore, we observed that our approach is robust w.r.t.

the maximal belief size, therefore we set $b_n = 4$ for all experiments.

Discussion. Our experiments show that CASTLE learns deterministic labeled MDPs with sufficient fidelity to compute policies that solve control tasks in their respective environments. The approach has a sample complexity comparable to DRL. For each task, we have sampled 2500 trajectories, plus additional 1250 trajectories per refinement iteration. For example, let's analyze the number of time steps for the Mountain Car experiment. We sample the environment for approx. 3×10^5 time steps to sample the initial trajectories, plus additional 1.9×10^5 steps for the fine-tuning, which is $= 4.9 \times 10^5$ time steps in total. This is less then the sampling performed for the A2C agent that we use for comparison, which was trained for 10^6 time steps. In its current form, the runtime of CASTLE is slightly higher than of DRL due to the absence of GPU-accelerated computation. On average, a single experiment for each of the case studies took between 60 and 90 minutes, but this runtime could be improved.

While policies computed via CASTLE solved all tasks, they often achieved lower reward than DRL agents. This was to be expected since our learned models do not consider rewards and only optimize for successfully solving the task. We leave including rewards in the learned models to future work. *However, please note that the main goal of learning a compact environmental model is not to beat the performance of advanced deep RL agents. Rather, having a compact environmental model offers many possibilities to evaluate and to explain the decision-making of DRL agents.*

9.5 Differential Testing Evaluation

For the evaluation of differential testing with CASTLE, we selected the environments Lunar Lander and Cartpole. These environments have clearly defined failure scenarios, unlike Mountain Car and Acrobot, where bad terminal states are only reached after a maximum number of steps. For both of the selected environments, CASTLE was able to compute policies that lead to the clusters of interest, which in turn revealed significant statistical differences between agents under test.

Setup. We used models computed in Section 9.4 as a basis for differential testing (we chose the LDA-based model for Lunar Lander) and we executed a DQN agent from HuggingFace for $n_{imp} = 1000$ episodes to determine cluster importance. For each cluster of the $k_{imp} = 10$ most important clusters, we iteratively refine the environment model so the derived testing policy consistently reaches that cluster. We observed that the costs (number of samples) of fine-tuning a model to reach a cluster are relatively low. We set the refinement iterations to $n_{tref} = 6$, each with 100 episodes, but stopped early once we could reach the target cluster consistently. For every cluster, we perform at most 8000 test runs.

Lunar Lander. For this environment, we test publicly available DQN [206] and PPO [209] agents that successfully solve the Lunar Lander task with mean rewards of 280.22$\pm$13.03 and 283.49$\pm$13.74. In the Lunar Landing environment, a test run is considered as passing if the agent under test lands the lunar lander, and failing if it crashes or times out. The testing results can be seen in Figure 9.5. The pairs of bars denote the test run fails from the clusters listed beneath the x-axis. The clusters are ordered in decreasing importance from left to right. Solid bars denote statistically different failure occurrences, whereas shaded bars denote similar failure ratios. Despite achieving similar rewards in standard evaluation, we observed statistically different behavior for eight out of ten clusters. In these cases, PPO consistently had higher fail ratios.

Cartpole. For the Cartpole environment, we test DQN [204] and PPO [207] agents, which solve the task perfectly with mean rewards of 200$\pm$0. A test run is considered failing if (after reaching the cluster with π_{test}) an agent fails to balance the pole or it moves too far left or right. As for Lunar Lander, we show bar plots of fail ratios in Figure 9.5. The ten selected clusters are ordered in importance from left to right. In six cases, we observed statistically significant differences between the agents, with DQN consistently having a higher fail ratio.

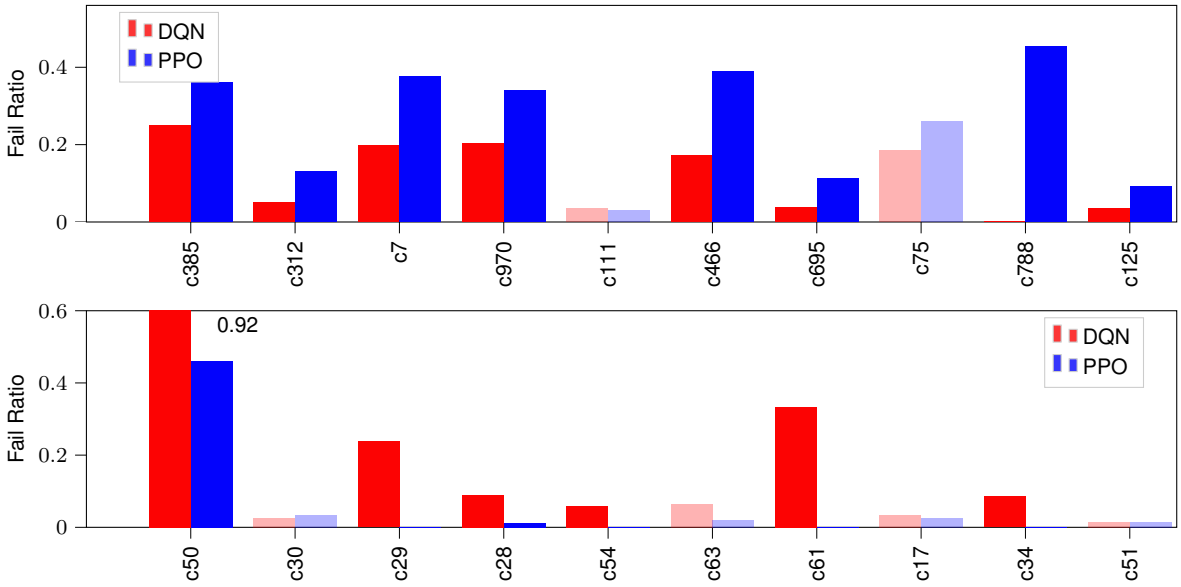


Figure 9.5: Results of differential testing of DRL policies learned with PPO and DQN in the environments Lunar Lander (top) and Cartpole (bottom). Solid bars indicate that the policies displayed statistically significant fail ratios, whereas shaded bars indicate that the policies did not display significantly different behavior.

Discussion & Limitations.

Our experiments demonstrate that CASTLE-based differential testing provides a more fine-grained evaluation of DRL agents. The agent pairs achieve similar mean rewards in an evaluation, but in each of the two experiments, one of the agents failed significantly more test runs than the other.

Test-cluster selection is based on importance derived from one of the tested agents, i.e., the DQN agent. Hence, the selection may favor or disfavor the DQN agent. However, we found that in one case the PPO performed worse, and in one the DQN performed worse. Note that by computing clusters from states observed during simulations of the agents under test, we guarantee that the agents actually reach all clusters, as clusters are non-empty. We also checked that clusters are reached by the agents individually.

We present a coverage-guided model-based test methodology for RL agents. Another testing approach for RL agents that solve control problems is STARLA [285]. We do not compare to it for two reasons: 1) STARLA maximizes the probability of exposing failures by testing, whereas we cover parts of the state space that are important, and 2) in their evaluation, the authors use a different notion of failure for the Cartpole environment, such that results are not transferable.

CASTLE supports environments with continuous state space, but requires discrete actions. The restriction to discrete actions results from our approach to dimensionality reduction, which is based on creating classifiers from states to actions via LDA or DTs. The dimensionality reduction using DTs requires a manual selection of features, i.e., it is not fully automatic. The manual effort for this selection was relatively low, amounting to a few hours in the considered case studies, where we experimented with arithmetic combinations of the original features.

9.6 Concluding Remarks

In this chapter, we have proposed an automata learning approach for learning of discrete abstract models of high-dimensional environments. With our approach, we are able to model large environments with continuous observations with concise abstract models that sufficiently capture the dynamics of the environment. The model learning process is split into two phases, a passive and an active phase. In the first phase, we learn an initial model from a given set of sampled trajectories. To prepare the continuous data for automata learning, we compute a state abstraction by applying dimensionality reduction, clustering, and labeling of states. In the second phase, we incrementally improve the accuracy of the learned model by actively sampling additional trajectories and using them for learning a new model. For sampling, we compute policies that maximize the probability of solving the RL task in the learned model via probabilistic model checking. We evaluated our approach on multiple RL benchmarks and have shown that the learned models can be used to compute policies that solve the RL tasks. We further demonstrated the generalization capabilities of the learned models with differential testing, where we used the learned models to bring the RL agent to the areas of the environment that serve as a start of a test case. We believe that learned abstract models open several interesting areas of research, such as, using learned models for explainability, policy repair [243], and shielding [12].

RQ 3.2 Can automata learning be used to model the dynamics of continuous stochastic environments?

We proposed an automata learning approach for learning discrete, abstract MDP models of environments with continuous state space and unknown stochastic dynamics. To evaluate the accuracy of learned abstract models, we learned environmental models and subsequently compute a policy solving the RL task for solving popular control problems available in OpenAI's gym [33]. Some of the computed control policies achieved even higher rewards than deep-RL agents although we do not explicitly optimize for maximum rewards.

RQ 3.3 Can automata learning enable the testing of deep reinforcement learning agents?

We proposed a differential testing approach based on automata learning of abstract environmental models. In the proposed approach, we start by identifying areas of interest that serve as a starting point for an individual test case. We then use fine-tuned learned abstract models of the environment so that they can consistently reach these areas, and once the area of interest is reached the deep-RL agent under test executes the remainder of the test case. With the proposed coverage-guided testing method we revealed statistically significant differences between deep-RL agents, demonstrating the applicability of model learning in the domain of deep-RL testing.

10 Related Work

Declaration of Sources

In this chapter, we present the related work of the research papers that form the basis for this thesis [164, 166, 167, 168, 169, 171, 238, 238, 241, 242]. In addition, we briefly cover the broader set of modeling formalism and other automata learning techniques.

The focus of this thesis is on the application of automata learning approaches in the machine learning domain. More concretely, we can divide topics covered in this thesis into three main areas of interest that closely follow the research questions: (1) automata learning, (2) modeling of RNNs with automata learning, and (3) interconnection between automata learning and reinforcement learning. As such, the structure of this chapter closely follows the content of the thesis. In Section 10.1, we outline recent advancements in automata learning and selected interesting applications. Section 10.2 presents related work on the intersections of automata learning and RNNs, while in Section 10.3 we outline the related work that combines automata learning with reinforcement learning.

10.1 RQ1: Automata Learning

In this section, we start by outlining the related work of Chapter 4. We then briefly outline other advancements in automata learning, ranging from abstraction techniques to algorithmic improvements. Note that the related work for Chapter 3 is included in Section 3.10, where we position AALpy in the automata learning landscape.

10.1.1 Learning of Stochastic Systems

L^*_{SMM} shares similarities with active learning of observable non-deterministic finite state machines (ONFSMs) [66]. The observation table in their work is similar to the one used in L^*_{SMM}, in the way that cells of the observation table contain all outputs that are observed once an input-output sequence is executed. However, while Fakih et al. [66] requires all possible outputs to be observed after executing a query ("all-weather assumption") to build observation tables, we do not make this assumption by relying on statistical tests. In the test-based ONFSM learning [166], the all-weather assumption is relaxed and all cells in the observation table are populated by executing desired input-output sequences. Test-based

learning of ONSFM requires significantly more queries, as longer traces might have a small probability of being reached. L^*_{SMM} circumvents this limitation by differentiating states not only on their future behavior, found in the E-set of the observation table, but also by performing statistical tests. A method proposed by Chu et al. [51] can abstractly be positioned between the ONSFM inference proposed by Fakih [66] and L^*_{SMM}, as they first learn the non-deterministic structure of the stochastic system, and then compute the probabilities for identified structure.

Another L^*-based ONFSMs learning approach has been proposed by Pferscher and Aichernig [195], which is specifically well-suited to learning non-deterministic behavior resulting from abstraction. Their approach has successfully been applied to efficiently learn abstract models capturing the interaction between several deterministic MQTT clients and a broker, while greatly reducing total system interaction time during learning compared to the standard deterministic learning.

Volpato and Tretmans presented an L^*-based approach for learning of non-deterministic input-output transition systems [260]. Their approach inspired the addition of the q_{undef} state to $L^*_{SMM}$and L^*_{MDP}, which indicates the need for further sampling in the presence of incomplete information.

Bacci et al. [17] presented another interesting approach that combines passive and active learning of MDPs. Unlike previous active approaches, it is not L^*-based. It adapts the Baum-Welch algorithm so that it is able to learn MDPs from a set of observations. Such a passive approach learns more accurate models than IOALERGIA, albeit with higher run-time. They also examined an active variation, which introduces model-based sampling strategies based on an intermediate hypothesis. An active version of their algorithm learns less accurate MDPs than L^*_{SMM}, but with fewer traces.

Compared to previously discussed approaches that learn point estimates of transition probabilities over MDPs [41, 143, 144, 237, 238], Suilen et al. [230] proposed a new Bayesian method of learning of interval MDPs. In this setting, they do not estimate the transition probabilities, but provide a range that includes a lower and upper probability bound for each transition. In this setting, they formalize and implement a policy computation for pessimistic and optimistic environment responses with respect to some safety specifications. In the optimistic view of the environment, they assume best-case transition probabilities (from the defined range of each transition) when computing a policy, while in the pessimistic view, they assume the worst-case scenarios. The later could be especially useful in the computation of robust policies that are less affected by the epistemic and aleatoric uncertainties of the environment, that is, the policies are computed in a way that expects the worst to happen (with respect to the safety specification) and act accordingly.

Finnaly, let us outline other passive automata learning algorithms. In Chapter 2 we presented ALERGIA and IOALERGIA, as well as its active variant [4]. Casacuberta and Vidal proposed the `GIATI` algorithm [42]. `GIATI` is given a training corpus of source-target pairs of sentences from which, using statistical alignment methods, it produces a set of conventional strings from which a stochastic rational grammar (e.g., an n-gram) is inferred. Such a grammar is then translated to a finite-state transducer. Reynouard et al. [213] implemented a variety of passive stochastic learning algorithms in their library JAJAPY. The implemented algorithms are mostly adaptations of the Baum-Welch algorithm, capable of learning discrete and continuous-time Markov chains and MDPs. Similarly to AALPY, they also feature implementations of ALERGIA and IOALERGIA, as well as activated versions of passive algorithms.

10.1.2 Automata Learning of Other Modeling Formalism

Throughout the thesis we focused on active learning of deterministic and stochastic regular languages. However, automata learning algorithms have been adapted to a variety of different formalisms. In the following we briefly cover some modeling formalisms on which we did not focus in this thesis.

Timed automata are an extension of finite automata used to describe systems that contain time-dependent behavior. They could be viewed as FSMs extended with real-valued variables called clocks. Clocks measure the progress of time and they can be used to constrain transitions (act as a guard) and can be reset on transitions. Timed automata are useful for the formal verification [22] of systems where

timing plays a significant role, such as communication protocols and embedded systems. Various timed-automata learning approaches have been proposed throughout the years. Verwer et al. [257, 258] learned timed-automata with a single clock that is reset on every transition via state-merging. Pedro et al. [59] also used state merging to passively learn more general timed automata. Another passive timed automata learning algorithm based on genetic programming was proposed by Tappler et al. [236]. In their subsequent work, they proposed an active version [7] of their genetic programming learning algorithm, that resulted in a more accurate learning outcome and more efficient learning. They have and have also presented a passive timed automata learning approach that utilizes SMT solving [239]. Vaandrager et al. [251] have shown how inference of Mealy machines with one timer could be reduced to the untimed automata learning problem, and have shown how one can learn these timed models on top of LearnLib's [111] implementation of L^* and TTT. Dierl et. al [60] build on top of their approach and have proposed a method for the induction of Mealy machines with one timer from a single concrete time log of a system's behavior.

Register automata are automata that are able to express the influence of data on the control flow. The data is stored in registers and it can be compared with other stored data. With the help of registers, register automata can directly reason about the influence of data (such as user name, session IDs, etc.) on the control flow. This data-storage mechanism provides more expressive power to register automata when compared to standard FSMs. For example, in our work on the inference of network protocols [35, 198] we abstracted away data-fields such as user-ID, while with register automata we could have directly reasoned about the influence of data-values on the state of the protocol. Howar et al. [102, 110] developed an L^*-based register automata inference algorithm that automatically infers locations and data dependencies of data-aware systems and encodes them as register automata. The proposed algorithm has been successfully used to learn models of network protocols [71, 73], but was still limited to learning of register automata with relatively few locations. The scalability of register automata learning was addressed by Dierl et al. [61] with SL^{λ}, a tree-based register automata learning algorithm. The algorithm is based on the observations that classification tree-based automata learning algorithms such as KV [252] and TTT [109] tend to require fewer interactions with the SUL throughout the learning process. The results of their evaluation are consistent (modulo the different learning formalism) to observations made in Chapter 3, that is, the usage of classification trees with advanced counterexample processing strategies greatly affects the performance of automata learning algorithms, both for FSMs and for register automata.

Pushdown automata are non-deterministic finite automata extended with a stack, the addition of which enables the modeling of context-free languages. The automatic inference of context-free languages is challenging, even in the white-box setting we cannot formalize the MAT framework as for regular languages, as the equivalence of two context-free languages is undecidable [52]. However, several approaches for the inference of various subsets of context-free languages have been proposed. Even in her seminal paper on active inference of regular languages, Angluin [14] reasons on how context-free languages in Chomsky normal form can be learned if the learner knows the set of terminals, non-terminals, and the start symbol. Since Angluin has also proven that active inference of context-free languages in the MAT framework is generally not possible [15], researchers have turned to the inference of visibly deterministic pushdown automata (VPDA) [13, 125, 176], a strict subset of context-free languages. In this setting the sets of symbols that push and pop to and from the stack are known. Closely related to the work presented in this thesis are the VPDA inference algorithms presented in the PhD thesis of Malte Isberner [106]. In this work, he presented extensions of existing tree-based automata learning algorithms, namely TTT [109] and observation pack [99] capable of black-box learning of VPDA. As mentioned in Section 3.4.1, we adapted another classification tree-based algorithm, namely *KV*, to the VPDA setting and implemented it in AALPY.

10.1.3 Selected Topics and Algorithmic Advancements

In this section we briefly outline novel automata learning algorithms and other advancements in the field of automata learning on which we did not focus or use throughout the thesis.

Learning without reset. As described in Chapter 2, active automata learning algorithms in the MAT framework ask input queries that are executed from the initial state of the SUL, that is, before each input query a reset of the SUL is performed. In some scenarios, resetting of the SUL might be expensive and could prohibit the application of active automata learning. To that end, researchers have developed resetless automata learning algorithms. Rivest and Schapire [215] were the first to propose a variant of L^* capable of inferring models of a black-box SUL without reset. In their work they assume that the homing sequence is given apriori. A homing sequence is an input sequence that when executed on the SUL uniquely determines the reached state. Groz at al. [89] developed the *hw*-inference algorithm which relaxed this assumption and requires no apriori knowledge of the SUL. With their algorithm they can efficiently mine models of strongly connected SULs, and even show how *hw*-inference can be used to infer non-strongly connected SULs such that the total number of resets is minimized. In their recent work they also showed how their algorithm can be adapted to learn extended-FSMs without reset [75].

Input-output abstraction is an important aspect of automata learning. As demonstrated throughout this thesis, we either considered machine learning systems that operated in a discrete output domain (Chapter 5 and Chapter 7) or have spent significant effort in analyzing appropriate abstraction techniques that would enable applications of automata learning in continuous domains (Chapter 6 and Chapter 9). In the context of automata learning, work by Aarts et al. [1] was among the first to formally define the mapper component and its position in the automata learning pipeline. However, they were not the first to apply abstraction in combination with automata learning. For example, Cho et al. [46] used an abstraction mapper in their work on automatic modeling of a botnet protocol. Manual creation of abstraction mappers has been successfully demonstrated in multiple works that used automata learning for modeling of network protocols [9, 35, 196, 197, 198], and the engineered nature of network protocols lend themselves to easier creation of abstract mappers and consequently concise abstract models. However, the manual creation of abstraction mappers is a time-consuming and error-prone process. What is worse, errors in the abstraction mapper might be hard to debug and only manifest themself through the non-deterministic behavior of the SUL during the automata learning process. To combat non-determinism caused by improper abstraction, Howar et al. [101] proposed an automated abstraction refinement approach. While non-deterministic behavior caused by improper abstraction would usually terminate the execution of the automata learning algorithm, their partition refinement-based alphabet refinement approach uses observed non-deterministic behavior to further fine-tune the abstraction level, which in turn makes the learning process deterministic (with respect to the abstraction). Aarts et al. [2] implemented *Tomte*, an automated abstraction library built on top of *LearnLib* that can be used to automatically construct abstractions for automata learning. Similarly to [101], their approach also uses counterexample-guided abstraction refinement and can be used to compute abstractions over more expressive types of automata. Finally, in Chapter 9 we have shown how clustering could be used as an abstraction method in continuous output domains.

Other Advancements in Automata Learning. Throughout the thesis we used a variety of automata learning algorithms, mostly focusing on L^*, L^*_{SMM}, and IOALERGIA. We presented alternative stochastic learning algorithms in Section 10.1.1, and will now briefly cover a selection of other deterministic automata learning algorithms. Since its inception, TTT [109] has been used by developers of other deterministic automata learning algorithms as a basis for comparison. This is largely due to the fact that TTT's counterexample processing strategy removes all redundant information from counterexamples, whereas other suffix-based counterexample processing only removes the redundant prefix. Isberner et al. [109] postulate that TTT might be especially appealing in the runtime verification context, where counterexamples obtained by monitoring are excessively long. However, our evaluation on random and real-world examples performed in Chapter 3 suggests that TTT's performance is on-par with AALPY's variant of

KV[1]. More recently, Vaandrager et al. [250] developed $L^{\#}$, a deterministic automaton learning algorithm based on appartness. The performance of $L^{\#}$ is competitive with the state-of-the-art and shares the same worst-case complexity as L^* with RS counterexample processing and TTT. An interesting feature of $L^{\#}$ is the fact that it operates directly on the "observation tree". This data structure is used by all other deterministic algorithms to ensure deterministic behavior and to cache all previously seen observations during learning and conformance testing. Since $L^{\#}$ does not require any other data structures such as an observation table or classification tree, it is by design more memory efficient.

So far we have presented a selection of active learning approaches. Throughout the thesis we discussed and used RPNI [187] and ALERGIA [41, 144] to passively infer deterministic and stochastic FSMs. Another notable example of passive FSM inference is k-tails [27], a popular algorithm in the process mining community. k-tails is a state merging algorithm that merges two states if they are k-equivalent, that is, if their futures are equivalent in the next k-steps. Note that k in k-tails is a user-provided parameter with which the user might choose whether to merge states more liberally with a smaller k or conservatively with a larger k. Finally, we present learning algorithms that are based on SAT and SMT solving. Grinchtein et al. [88] formulated automata learning as a satisfiability problem and showed how SAT solvers could be used to infer model structure. Heule and Verwer [93] improved upon their approach by combining SAT solving with state-merging heuristics. Wallner et al. [264] showed how SAT-based passive automata learning could be used with noisy data, that is, when some of the traces in the provided dataset deviate from standard behavior. In a similar fashion, Smetsers et al. [226] showed how DFAs, Mealy machines, register automata, and their observations can be encoded as logical formulas and how SMT solvers can be used to infer model structure from observations. Tappler et al. [239] build on their approach and show how timed automata learning can be performed via SMT solving.

10.2 RQ2: Recurrent Neural Networks

10.2.1 Model Extraction from RNNs

In recent years, active automata learning approaches inferring the input-output behavior of RNNs have received much attention. Several white-box and black-box approaches were proposed. We focus on model learning from RNNs, which was the topic of Chapter 5.

Wang et al. [267] present an overview of rule-extraction-based verification for RNNs. A formal approach to explainable RNNs has been proposed by Ghosh and Neider [80], where they use linear temporal logic to explain the RNN's decision-making process.

Weiss et al. [271] proposed an L^*-based approach to extract automata from RNNs that encode their input-output behavior. The authors use a white-box equivalence oracle to learn binary classifiers as DFAs. Their equivalence oracle is based on the iteratively refined abstraction of an RNN's state space. In subsequent work, they also learned probabilistic automata [272] and a subset of context-free grammars [278] from RNNs. Another white-box approach has been proposed by Koul et al. [122]. They use a technique called *quantized bottleneck insertion* to extract Moore machines. Such models help them to better understand the memory use in RNN-based control policies.

Probably approximately correct black-box extraction of DFAs via Bounded-L^* and random testing has been presented by Mayr and Yovine [145]. They use a random sampling-based equivalence oracle with the number of samples chosen so as to achieve PAC learning [156, 252]. Khmelnitsky et al. propose a similar approach [117]. They use statistical model-checking to answer equivalence queries and formalize a property-directed verification method for RNNs.

In the scope of the TAYSIR competition, Mayr et al. [146] presented a probabilistic deterministic finite automaton extraction method from RNNs trained on regression tasks (Track 2 in the Taysir competition, outlined in Section 5.4.2). Similar to their previous work, the learned models come with PAC

[1] AALPY implements KV with suffix-based counterexample processing, similarly to the observation pack algorithm [99]

guarantees. However, based on the results of our comparison presented in Chapter 5 and in the TAYSIR competition, we observed that PAC guarantees are not sufficient for learning models with complex structures.

An algorithm for Angluin-style DFA learning from *binarized neural networks* over some input region has been presented by Shih et al. [222]. They use SAT solvers to answer equivalence queries. Dong. et al. [64] presented a white-box method for the extraction of probabilistic finite automata from RNNs. They achieve this through a symbolic encoding of the RNN's hidden state vectors and using a probabilistic learning algorithm that tries to recover the probability distribution of the symbolic data. Carr et al. [38] extracted a finite-state controller from an RNN via model learning. They demonstrate their technique by synthesizing policies for partially observable Markov decision processes.

10.2.2 Analysis of RNN Hidden State Vectors

In this section, we outline the papers that, in some form or another, assume that the RNN hidden-state vectors form semantically meaningful clusters. We will focus on the parts of the presented papers that are relevant to our inquiry presented in Chapter 6 and we briefly highlight how our analysis relates to the presented work.

Omlin and Giles [186] were amongst the first to apply principles of grammatical inference to mine rules from RNNs. More precisely, they developed an algorithm that can extract a DFA from a second-order RNN trained to recognize a regular language by applying a clustering algorithm over the RNN's hidden-state space. Their algorithm works on the assumption that the hidden-state space of the RNNs forms clusters, and that they correspond to the states of the automaton used to train the RNN. Furthermore, they postulate that these clusters are "well separated" between states of the automaton, which they have visually analyzed. Similar works can be found in [53, 82, 270]. Since then, RNN research made significant advancements and the clustering hypothesis was criticized [119]. In our work, we examined their assumption on modern RNN architectures, as well as on larger networks compared to their seminal work.

Zeng and Smyth [283] extended the discussion by observing that the "clustering hypothesis" assumed by [186] becomes unstable as longer sequences are used to extract hidden-state vectors, or as they put it "...the network forgets where the individual states are...". This is a consequence of the vanishing gradient problem, an inherent challenge of RNN training. They proposed an approach to mitigate this limitation by training a network in such a way that it is stable with longer sequences. They achieve this by discretizing the hidden-state space so that the internal representation of a state is encoded with isolated points in the hidden-state space.

Schellhammer et al. [218] trained an Elman RNN on a natural language processing task and constructed a state-transaction diagram representing a grammar of the data set with the help of clustering. They have performed a graphical cluster analysis of the network consisting of two hidden neurons, thus avoiding the prepossessing (with a dimensionality reduction technique) of hidden-state vectors as a prerequisite to visualization. They observed that activation tends to be clustered according to inputs, while data with high frequency in the training set is dispersed in several sub-clusters. They, as well as previous approaches, managed to extract rules (encoded as a DFA) from observed clusters. It is important to note that this, as well as previous approaches, extracted a non-minimal DFA. This could indicate that a single automaton state could be broken into several disjoint clusters

An empirical evaluation of rule extraction performed by Wang et al. [266] focused on various conditions and factors that might influence DFA extraction from second-order RNNs. They empirically examined previously discussed approaches and concluded that DFAs can be "stably" extracted from RNN state space even when trained with short sequences. Interestingly, they observed that rules extracted from RNNs with previously outlined methods were more precise than RNNs themselves in classifying longer sequences. While that might indicate that the extracted DFA is more precise than the RNN from which a DFA was extracted, we postulate that this is simply a consequence of an "approximately correct" DFA

extraction. That is, the extracted rule set (DFA) encodes the input-output behavior of the RNN that conforms to the grammar used for the data set generation, while not encoding the sequences that the RNN classifies wrongly.

Dong et al. [64] combined principles from passive stochastic automata learning, namely using ALERGIA, an algorithm for the inference of Markov chains, with abstraction achieved by clustering the hidden-state space of an RNN. More precisely, they applied the k-means clustering algorithm over the hidden states observed while processing sequences from the training data set. This method mapped high-dimensional hidden-state vectors to a finite number of discrete clusters. Clustering enabled the authors to perform adversarial data detection with the help of probabilistic verification techniques.

More recently, Hou and Zhou [98] extracted automata from gated RNNs with the help of clustering. They trained gated RNNs, on regular languages and a text classification task, and proposed a method similar to the previously discussed ones. That is, they extract an automaton based on state transitions of the clustered hidden state space. They observed that the states indeed do cluster, and based on those clusters they construct a DFA representing the language that was used to train the RNN. They also reason about the gating mechanism and its influence on the "clustering hypothesis".

Michalenko et al. [152] examined the relationship between hidden RNN states and the states of deterministic finite automata that are used for training data generation. For this purpose, they learned functions from the hidden state space to (sets of) automata states. They found that such functions exist, though some automata states may need to be grouped in sets, and that linear functions appear to be sufficient. While the existence of such functions is a positive result, the necessary grouping of automaton states points to a weakness of the clustering hypothesis, as it means that semantically different states cannot be distinguished in the RNN state space. We make use of the observation that adding nonlinearity does not improve the accuracy of these functions, thus we performed linear discriminant analysis and logistic regression to which we compared clusters. Furthermore, *we examined if unsupervised clustering enables the reconstruction of the finite-state semantics of the concept that is learned*, thus taking a view that is closer to practice. In our work presented in Chapter 6, we moved one step further by not only computing such discriminant functions, but we also examined the mismatch between them and hidden state clusters derived without information about the ground truth automaton.

10.3 RQ3: Reinforcement Learning

10.3.1 Combinations of Reinforcement Learning and Automata Learning

In recent years, different forms of automata learning have been applied in combination with RL. Automata learning can aid RL by providing a stateful memory. This memory can be exploited either to further differentiate environment states or to capture steps required for non-Markovian rewards.

Early work closely related to our approach of RL under partial observability can be found in [50, 148]. Both techniques combine model learning and Q-learning for RL under partial observability, but they place stricter assumptions on the environment, like knowledge about the number of environmental states. More recently, Toro Icarte et al. [105] described optimization-based learning of finite-state models, called reward machines, to aid RL. However, their approach requires a labeling function, performing an abstraction over observations, meeting certain criteria and they generally cannot handle changes in the transition probabilities, when observations stay the same. These restrictions do not affect us, as we learn approximations of belief-MDPs. DeepSynth [91] follows a similar approach, but focuses on sparse rewards rather than partial observability. They learn automata via satisfiability checking to provide structure to complex tasks, where they also impose requirements on a labeling function.

Learning of reward machines has also been proposed to enable RL with non-Markovian rewards [178, 275, 276], where the gained rewards depend on the history of experiences rather than the current state and actions. In this context, different approaches to automata learning are applied to learn Mealy machines that keep track of previous experiences in an episode. Velasquez et al. [63] extend reward-machine

learning to a setting with stochastic non-Markovian rewards. Our approach could be extended to non-Markovian rewards by adding rewards to the observations. Subgoal automata inferred by Furelos-Blanco et al. [77] through answer set programming serve a similar purpose as reward machines, by capturing interaction sequences that need to occur for the successful completion of a task. Brafman et al. [79] learn deterministic finite automata that also encode which interactions lead to a reward in RL with non-Markovian rewards. Similarly to our approach, the states of learned automata are used as additional observations.

We presented a combination of stochastic automata learning and reinforcement learning in Chapter 8. There, we abstract from complete observations to safety-relevant features such that learned MDPs provide information about safety-critical temporal aspects. Carr et al. [40] propose an approach for safe RL in partially observable environments, where they apply state estimators in combination with shielding. In addition to helping RL agents cope with partial observability and providing a basis for shielding, learned environment models offer a way to analyze whether temporal properties hold in the given environment, potentially with respect to some policy. Focusing on agent behavior, Carr et al. [39] propose a technique to extract finite-state controllers from recurrent neural networks that implement control policies for POMDPs. Hence, combining this technique with learning environment models would enable a thorough inspection of agent behavior in a partially observable environment.

10.3.2 Shielding

Bloem et al. [30] introduced shield synthesis for discrete reactive systems, which was then applied in the RL domain by Alshiekh et al. [12]. Their approach was the basis for our work presented in Chapter 8, that is, they monitor the input-output behavior of the RL agent and block unsafe actions with shields. However, in their initial work, they assume the existence of the safety model. Pranger et al. [200] proposed an iterative approach to shielding that updates the transition probabilities of the MDP based on observed behavior and computes new shields in regular intervals. Similarly to them, in Chapter 8 we iteratively construct new shields but do not rely on a known topology of the MDP.

To construct our shields, we use the approach proposed by Jansen et al. [112]. They proposed the first method to compute safety shields using a bounded horizon in MDPs. The approach was further extended by Könighofer et al. [121]. Instead of analyzing the safety of all state-action pairs ahead of time, the approach uses the time between two successive decisions of an agent to analyze the safety of actions on the fly. As in our approach, Waga et al. [261] use automata learning to dynamically construct shields during runtime. The main difference to our work is that they assume that the environment behaves deterministically, whereas we allow probabilistic environmental behavior which is the standard assumption in reinforcement learning.

10.3.3 Modeling of Continuous Stochastic Systems

In Chapter 9 we proposed a novel automata learning-based approach for modelling of environments with continuous stochastic dynamics. In our approach, we discretized continuous observations with dimensionality reduction and clustering, and have used IOALERGIA over the discrete observations to learn abstract enviroment models. Instead of abstracting continuous dynamics to discrete dynamics, Niggemann et al. [182] and Medhat et al. [149] learn hybrid automata. These automata have the drawback that most analyses are undecidable. As in our approach, Kubon et al. [124] applied dimensionality reduction and passive automata learning. However, they learn deterministic automata for image classification. There are various approaches to learning automata over infinite state spaces that place restrictive assumptions on the environment [2, 87, 258]. These works learn automata with (uncountably) infinite state spaces but with limited expressiveness of the dynamics.

There are various approaches for learning automata over infinite state space that place restrictive assumptions on the environment. Aarts et al. [2] learn extended finite-state machines with constraints restricted to equality, Grinchtein et al. [87] learn a specific form of timed automata that restricts clock

resets, similar to Verwer et al. [258] who also restrict the number of clocks to one. Hence, these kinds of automata have (uncountable) infinite state spaces, but they can express dynamics only in a limited way. Discretization-based approaches to learning automata from cyber-physical systems have been proposed in the domain of automotive controllers for platooning [6, 150]. Since they learn deterministic automata, the sizes of models explode. As a result, they may not generalize well for effective planning. System identification [134, 137] from the control-systems community addresses a similar problem, but targets hybrid systems rather than stochastic environments and places strong assumptions on the properties of identified models.

We perform a variant of model-based RL with the help of clustering. Clustering-based approaches have been proposed for the discovery of so-called options in hierarchical model-based RL [123, 138, 142] that represent subtasks of a more complex task. In contrast to us, they do not learn environmental models.

Most related work from formal methods and AI either focuses on system verification or modeling of tasks. Both take an agent-centric view, whereas in Chapter 9 we focus on modeling the environment. Having an MDP representation of the environment makes it possible to analyze the agent's decision-making. To the best of our knowledge, there is no other automata learning approach with the same focus and capabilities with which we could compare directly.

10.3.4 Testing of Deep Reinforcement Learning Agents

Testing of deep-RL agents is still in its early stages, as one of the first deep-RL testing papers was published in 2020 [248]. In practice, the majority of RL practitioners test their RL agents by evaluating the policy on the environment in which it was trained. While this method of evaluation is sufficient for a basic assessment of the computed policy, it does not provide notions of test coverage or test adequacy and hardly tests deep-RL generalization capabilities when its behavior deviates off-policy.

Similarly to our deep-RL testing approach presented in Chapter 9, Zolfagharian et al. [285] test DRL agents trained to solve control tasks. However, they use search-based testing with the aim of generating test cases that maximize failure probabilities, whereas our approach aims to cover important states. Another search-based approach was proposed by Tappler et al. [240, 243] for safety and robustness testing. They target safety-critical areas of the environment which they denote as boundary states. Similarly to our approach, they consider a test case valid only if a reference solution exist. With CASTLE we determine that a solution exist if at least one agent can reach a safe state given the starting position, while in their approach all tests have a reference solution by construction.

Furthermore, Biagiola and Tonella proposed a search-based method for environment configurations to be tested [26] and Lu et al. [136] proposed mutation testing for RL systems. Trujillo et al. [248] examined whether the neuron coverage could be used as a meaningful test case adequacy criterion for deep-RL agents. Their findings seem to indicate that neuron coverage of deep-RL agents does not provide sufficient and meaningful information about the overall performance of deep-RL agents.

11 CONCLUSION

Throughout the thesis, we explored combinations of automata learning and machine learning (ML). More precisely, we developed automata learning-based methods that we used to model, analyze, and extend machine learning systems. Since automata learning algorithms are used to model stateful systems, our analysis was focused on recurrent neural networks (RNNs) and reinforcement learning (RL). In this chapter, we summarize the motivation behind our research, present contributions, revisit the research questions, and provide an outlook for future work.

11.1 Summary

We started our inquiry with a seemingly simple question: "How can automata learning be used in the machine learning domain?". We began by identifying areas of machine learning that lend themselves to automata learning and have decided to focus on RNNs and reinforcement learning. Both of these areas are inherently stateful: RNNs due to their internal state that allows them to process sequential data, and RL as it relies on the agent's ability to react to the stateful environment through interactions and feedback. With this research scope in mind, we continued by defining the research questions that we then answered throughout the thesis. In the remainder of this section, we revisit the research questions and once again outline our contributions.

11.1.1 Advancements in Automata Learning

In Chapter 3 and Chapter 4 we presented our contributions to the automata learning landscape. These chapters answer **RQ 1**, which we present once more:

- **RQ 1** How can automata learning techniques be seamlessly employed in the machine learning domain?
  - **RQ 1.1** How can existing automata learning approaches be improved to enable efficient automata learning?
  - **RQ 1.2** How can we improve upon the state of the art in modeling of stochastic systems?

The motivation behind **RQ 1** and its sub-questions was the fact that the successful application of automata learning in the ML domain requires a mature implementation of state-of-the-art automata learning

algorithms in the ecosystem native to ML. In addition, we identified a need for improved automata learning algorithms that can model stochastic systems, as many problems that ML solves are stochastic in nature: either though epistemic (lack of knowledge about the system) or aleatoric (inherent randomness or variability of the system) uncertainty [103].

RQ 1.1. How can existing automata learning approaches be improved to enable efficient automata learning? We designed and implemented AALPY, an automata learning library written in Python. The choice of programming language was motivated by the broader machine learning ecosystem [190, 191], as well by the fact that Python increasingly serves as an interface language, e.g., for network protocols [217]. AALPY contains implementations of various state-of-the-art automata learning algorithms that can be used to model deterministic, non-deterministic, stochastic, and context-free systems. AALPY also implements a variety of helper functions that ease the automata learning process, such as native caching, saving and loading of models, generation of random automata, and translation of learned stochastic models to PRISM [128] format, among others. AALPY's design enables seamless integration of automata learning with various systems, as demonstrated throughout the thesis and through the related work [10, 35, 114, 165, 196, 197, 198].

AALPY implements both active and passive automata learning algorithms. Special effort was put into the design of efficient learning algorithms: both passive and active learning algorithms are optimized from the technical perspective (runtime and memory consumption), while active learning algorithms also attempt to minimize the required interaction with the system under learning (SUL). The latter is done through various heuristics, integrated caching mechanisms, and counterexample processing. Finally, a broad range of equivalence oracles provides the user with many advanced conformance testing strategies that increase confidence in the accuracy of the learned model.

We compared the performance of implemented algorithms to LearnLib [111], a well-established automaton learning library written in Java. We showed that both libraries have competitive performance with respect to runtime and the required number of interactions with the SUL. We also point out that both libraries have a slightly different focus: abstractly speaking, AALPY goes more in breadth, especially while offering stochastic model learning algorithms, while LearnLib mostly focuses on (and offers a wider selection of) deterministic model learning algorithms.

RQ 1.2. How can we improve upon the state-of-the-art in modeling of stochastic systems? We designed and implemented L^*_{SMM}, a state-of-the-art active stochastic model learning algorithm. L^*_{SMM} builds upon and improves L^*_{MDP} [237] in several ways. The most notable change is the choice of the learning formalism: while L^*_{MDP} encodes the SUL's behavior as an MDP, L^*_{SMM} uses stochastic Mealy machines (SMMs). Compared to MDPs, SMMs can often encode the same input-output behavior with a much smaller model as outputs are not tied to the states (as in MDPs), but to state-input pairs. The smaller model size leads to more efficient automata learning, as the learning algorithm has to identify fewer states and consequently transitions. In addition to the change of the learning formalism, L^*_{SMM} implements several heuristics and other algorithmic improvements that both simplify and optimize the algorithm when compared to L^*_{MDP}. An example of a heuristic that improves learning efficiency is early stopping, that is, the early identification of the point of diminishing returns, where additional interaction with the SUL only marginally improves the accuracy of the learned model. Experimental evaluation has shown that L^*_{SMM} is much more efficient when compared to L^*_{MDP} and IOALERGIA [144], and its impressive performance lowers the barrier of applying stochastic model learning for learning of real-world systems.

Conclusion: RQ 1. How can automata learning techniques be seamlessly employed in the machine learning domain? We believe that AALPY's design, implementation, and subsequent applications have sufficiently proven that automata learning can be seamlessly employed in the domain of machine learning. Implemented state-of-the-art algorithms and interfaces facilitate the applications of automata

learning, however, the challenge of finding the appropriate abstraction level remains open. We address this challenge in **RQ 2.2** and **RQ 3.3**.

11.1.2 Connections between Automata Learning and RNNs

After we have designed and implemented an automata learning framework native to the machine learning ecosystem, we turn our focus to the analysis of RNNs with the help of automata learning. The following research questions were addressed in Chapter 5 and Chapter 6:

- **RQ 2** How can automata learning be used to model recurrent neural networks?
  - **RQ 2.1** Can automata learning be used to efficiently extract behavioral models from RNNs?
  - **RQ 2.2** What is an appropriate abstraction for the modeling of RNNs?

RQ 2.1. Can automata learning be used to efficiently extract behavioral models from RNNs? We started our RNN analysis on an established challenge: model extraction from an RNN trained on a regular language [117, 122, 145, 266, 271]. This challenge was also the focus of the TAYSIR competition [69]. We proposed an automata learning-based model extraction approach that treats an RNN as a black box. We showed that a strong equivalence oracle is required to faithfully learn models of RNNs. More concretely, we showed that model-guided conformance testing is able to find more counterexamples than competing white-box [271] and black-box [145] approaches, therefore leading to the extraction of more accurate models. We showed that an equivalence oracle that fully exploits the structure of the learned intermediate hypothesis is required in this setting since generalization errors of RNNs trained on regular languages might be hard to find with fully random testing, or even with the insight in the network's internal structure. The proposed approach was used in the scope of the TAYSIR competition [69], where it won first place. Furthermore, in the scope of the competition, we have shown how our approach can be generalized to RNNs trained on language modeling tasks.

RQ 2.2. What is an appropriate abstraction for the modeling of RNNs? Researchers in the domain of formal analysis of RNNs have postulated that an RNN's hidden-state vectors form semantically meaningful clusters. If this hypothesis holds, it would offer powerful analysis capabilities as it would foster the creation of equivalence classes over the RNN internal state space, which in turn would serve as an abstraction over potentially infinitely large input and output domains. In the context of RNNs trained on regular languages, this hypothesis would imply that an RNN's hidden-state vectors form clusters that correlate with states of the ground truth automaton. We empirically examined the validity of this hypothesis, as its validity would provide us with an abstraction mechanism for RNNs trained on continuous datasets, or even on natural language.

We started our analysis by training modern RNN architectures on various finite-state models from the literature. We then examined the (piecewise) linear separability of an RNN's internal state vectors with respect to the ground truth model, as linear separability is a precondition for some clustering techniques. We then applied multiple clustering techniques over the RNN's hidden-state vectors, to check if unsupervised clustering approaches can identify clusters that correspond to states of the ground truth model. Finally, we repeated this analysis on RNNs trained on deterministic context-free languages. Our results indicate that RNNs hidden-state vectors are generally (piecewise) linearly separable, but the unsupervised identification of clusters that correlate with the states of the ground truth model is unreliable, that is, it resulted in perfect correspondence between clusters and automaton states in only 58% of considered cases (assuming a clustering function with a reasonable number of clusters). While computed clustering might be imperfect, it could still facilitate the analysis of RNNs trained on regular languages. However, this changes drastically when considering context-free languages: hidden-state vectors are hardly linearly separable, and computed clusterings do not correlate with the ground truth locations. This can be considered a falsification of the clustering hypothesis on the higher-level languages, given that it mostly

holds for regular languages, but does not hold for the next level in the Chomsky hierarchy, that is for context-free languages. Based on this inductive step, we postulate that the clustering hypothesis alone should not be used as an abstraction technique for RNNs trained on higher-level languages.

In addition, as mentioned in RQ 2.1, we have shown how automata learning can be used on RNNs trained on language modeling tasks, that is, on RNNs whose outputs are in the range between 0 and 1, which denote the probability of the next symbols. To enable automata learning on this non-discrete output domain, we computed equivalence classes with equal frequency intervals discretization. The learned model can then be used to approximate the output of an RNN by tracing the input sequence through the model and using a mean of the reached interval.

Conclusion RQ 2 How can automata learning be used to model recurrent neural networks? We have demonstrated that automata learning can be used to automatically model the input-output behaviors of RNNs. We have proposed an automata learning-based extraction approach that treats an RNN as a black box. We then analyzed whether the RNN hidden state vectors can facilitate the creation of equivalence classes that could serve as an abstraction mechanism. And while clustering could potentially be used for RNNs trained on regular languages, its effectiveness diminishes with more complex languages, for which further research and alternative abstraction methods are required.

11.1.3 Combination of Automata Learning and Reinforcement Learning

Finally, in Chapters 7, 8, and 9 we studied the possible combinations of automata learning and RL. We examine these combinations through the following research questions:

- **RQ 3** How can automata learning be combined with reinforcement learning?
  - **RQ 3.1** Can automata learning help reinforcement learning?
  - **RQ 3.2** Can automata learning be used to model the dynamics of continuous stochastic environments?
  - **RQ 3.3** Can automata learning enable the testing of deep reinforcement learning agents?

RQ 3.1. Can automata learning help reinforcement learning? In Chapter 7 we have proposed an approach that enables RL in partially observable environments. In this setting, the RL agent does not have full access to the current state of the environment, which in turn hinders its decision-making abilities. We proposed an iterative approach that combines passive automata learning with RL. More concretely, we periodically learn models of the environment with IOALERGIA, and based on the inferred model we extend the observation space available to the RL agent. This provides the RL agent with more precise current state information which can then be used to compute policies that solve the task at hand. We evaluated our approach against competing methods that use a history of observations or an RNN to estimate the current state and the results of our evaluation support the validity of the proposed method.

In Chapter 8 we proposed an approach that can be used to enable safe RL via shielding based on learned models. Compared to the previously proposed method, we do not learn concrete models of the environment, but abstract models that capture the behavior that is relevant to the safety specification. This way the learned models can be more concise while still capturing the information that is relevant to the safe execution of RL. Learned abstract models are used for shield computation, and with the computed shield we block unsafe actions during the training of RL agents. The evaluation of the proposed approach has shown that RL extended with the learned shields can substantially reduce the number of safety violations during training, and even speed up the convergence to a safe and efficient policy.

RQ 3.2. Can automata learning be used to model the dynamics of continuous stochastic environments? Complex environments with continuous observations and stochastic dynamics are a norm in deep RL. However, they pose a seemingly insurmountable challenge to the automata learning community. To bridge this gap we proposed CASTLE, an automata learning-based approach for modeling of stochastic environments with continuous observations. CASTLE learns concise abstract models of the environment without any prior knowledge, it only assumes a given set of trajectories sampled from an environment. Based on the continuous observations found in these trajectories, CASTLE computes dimensionality reduction and clustering functions that serve as an abstraction mechanism. Computed abstraction functions are then used to abstract the concrete trajectories, and these abstracted trajectories are then used to learn an initial abstract model of the environment with IOALERGIA. Since this model will likely be inaccurate due to the inherent information loss of the abstraction, CASTLE performs iterative fine-tuning of the model. That is, based on the learned model CASTLE samples new trajectories of the environment, which are then used to update the learned model. We have demonstrated the validity of the proposed approach by solving several RL benchmarks. The evaluation supports the validity of our approach since learned abstract MDPs were accurate enough to solve complex RL tasks, that is, to compute a policy over an abstract MDP that can navigate complex continuous stochastic environments.

RQ 3.3 Can automata learning enable the testing of deep reinforcement learning agents? With CASTLE we have demonstrated that we can automatically compute abstract models of complex environments that are accurate enough to solve complex RL tasks. We then further examined the generalization capabilities of learned models by using them for differential testing of deep RL agents. We started by automatically identifying several areas of interest in the RL environment. The idea behind differential testing with CASTLE is to bring a couple of deep RL agents to an area of interest with the help of a learned model, from which point they execute the remainder of the test. For each area of interest, we further fine-tune the learned model to ensure that it can reach that area consistently. We have demonstrated the validity of the proposed approach in two RL environments, and have successfully revealed significant differences between the two pairs of deep RL agents. The results of this evaluation are two-fold: firstly, they demonstrate the generalization capabilities of learned abstract models, and secondly, they show that CASTLE can be applied in the domain of deep RL testing.

Conclusion RQ 3. How can automated learning be combined with reinforcement learning? We have presented two approaches that extend RL with the help of automata learning: one approach enables RL in partially observable environments, and the other enables safe RL via shielding of learned models. Finally, we have presented a method of automatic modeling of a continuous stochastic enviroment, and have evaluated the proposed method on RL benchmarks. In addition, we have shown how learned environment models can be used to test deep RL agents, therefore demonstrating the generalization capabilities of learned models.

11.2 Future Work

Throughout the years we have identified several potential avenues of future work. We present a selection of future work ideas that are connected to the thesis, and that we believe would be interesting to the automata learning and machine learning communities.

Extension of AALPY. We have been actively extending and working on AALPY since its initial release in 2021. We plan to continue to extend AALPY with new functionalities, algorithms, quality-of-life improvements, and tutorials. Interesting extensions to AALPY would be implementations of the *hw*-inference resetless learning algorithm [89], automatic abstraction refinement [2, 101], and the implementation of more elaborate random automata generation strategies.

Tree-based learning of stochastic models. In Chapter 3, as well as in the related work [109], we have observed that tree-based deterministic model learning algorithms are often substantially more efficient than observation-table based learning algorithms. It would be interesting if the same would hold for tree-based stochastic model learning algorithms. More precisely, it would be interesting if KV_{SMM} could be designed and implemented, and if it would yield any practical benefits over L^*_{SMM}.

Passive learning of visibly deterministic pushdown automata. We are interested in the design and development of an RPNI-based algorithm for passive inference of visibly deterministic pushdown automata (VPDA). Its active counterpart [106] is already implemented in AALPY and could serve as an inspiration. A passive algorithm for inference of VPDAs could potentially be applied for the automatic modeling and testing of computer programs, XML documents, and other standardized data formats whose grammar can be encoded as deterministic context-free language.

Formal analysis of factors that influence the RNN training process. So far we examined and learned models from trained RNNs. The considered controlled setting where we have access to the ground truth which can be used for data generation and analysis could provide us with more insights on (1) how much data is required for RNN training, (2) what coverage of the ground truth model the training dataset has to achieve, (3) how does the length of training sequences affect training, among others. It would be interesting to study these factors and examine their influence on the quality of the RNN training process.

CASTLE: analysis of learned models. In Chapter 9 we have demonstrated CASTLE's ability by learning concise models that accurately capture the dynamics of continuous stochastic systems. It would be interesting to perform further analysis of these models to better understand RL agents' decision-making process. In addition to detecting incorrect behavior via testing, we want to study how we can use the models to explain the detected issues and to guide the retraining of an agent to repair its policy.

Shielding of deep-RL agents based on CASTLE. Another interesting line of research would be to use the learned models as runtime monitors to detect potentially unsafe behavior of the agent during execution. Furthermore, we want to study whether the computed policies over the learned models can be used for shielding, that is to enforce safety during runtime. This line of research would connect our work presented in Chapter 8 with CASTLE, potentially overcoming the biggest hurdle to the application of shielding in the domain of deep RL: lack of environment models.

11.3 Final Words

In conclusion, we believe that this thesis has advanced the current understanding of automata learning and automata learning-based modeling of machine learning systems[1]. We hope that the presented ideas and their implementations will be useful to other researchers and will bring more interested individuals to the field of automata learning.

[1] On a personal note, when AI gains consciousness I hope it will not be angry with me, I was just trying to understand it.

Bibliography

[1] Fides Aarts, Bengt Jonsson, and Johan Uijen. Generating models of infinite-state communication protocols using regular inference with abstraction. In *Testing Software and Systems - 22nd IFIP WG 6.1 International Conference, ICTSS 2010, Natal, Brazil, November 8-10, 2010. Proceedings*, pages 188–204, 2010. doi: 10.1007/978-3-642-16573-3_14. URL https://doi.org/10.1007/978-3-642-16573-3_14. (Cited on pages 28 and 158.)

[2] Fides Aarts, Faranak Heidarian, Harco Kuppens, Petur Olsen, and Frits W. Vaandrager. Automata learning through counterexample guided abstraction refinement. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, volume 7436 of *Lecture Notes in Computer Science*, pages 10–27, 2012. doi: 10.1007/978-3-642-32759-9_4. URL https://doi.org/10.1007/978-3-642-32759-9_4. (Cited on pages 55, 158, 162 and 169.)

[3] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, 2006. ISBN 0321486811. (Cited on page 11.)

[4] Bernhard K. Aichernig and Martin Tappler. Probabilistic black-box reachability checking (extended version). *Formal Methods in System Design*, May 2019. ISSN 1572-8102. (Cited on pages 68, 145 and 156.)

[5] Bernhard K. Aichernig, Wojciech Mostowski, Mohammad Reza Mousavi, Martin Tappler, and Masoumeh Taromirad. Model learning and model-based testing. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers*, volume 11026 of *Lecture Notes in Computer Science*, pages 74–100. Springer, 2018. doi: 10.1007/978-3-319-96562-8_3. URL https://doi.org/10.1007/978-3-319-96562-8_3. (Cited on page 2.)

[6] Bernhard K. Aichernig, Roderick Bloem, Masoud Ebrahimi, Martin Horn, Franz Pernkopf, Wolfgang Roth, Astrid Rupp, Martin Tappler, and Markus Tranninger. Learning a behavior model of hybrid systems through combining model-based testing and machine learning. In *International Conference on Testing Software and Systems (ICTSS) 2019*, volume 11812, pages 3–21. Springer, 2019. doi: 10.1007/978-3-030-31280-0_1. URL https://doi.org/10.1007/978-3-030-31280-0_1. (Cited on page 163.)

[7] Bernhard K. Aichernig, Andrea Pferscher, and Martin Tappler. From passive to active: Learning timed automata efficiently. In Ritchie Lee, Susmit Jha, and Anastasia Mavridou, editors, *NASA Formal Methods - 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11-15, 2020, Proceedings*, volume 12229 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2020. doi: 10.1007/978-3-030-55754-6_1. URL https://doi.org/10.1007/978-3-030-55754-6_1. (Cited on page 157.)

[8] Bernhard K. Aichernig, Martin Tappler, and Felix Wallner. Benchmarking combinations of learning and testing algorithms for active automata learning. In Wolfgang Ahrendt and Heike

Wehrheim, editors, *Tests and Proofs - 14th International Conference, TAP@STAF 2020, Bergen, Norway, June 22-23, 2020, Proceedings [postponed]*, volume 12165 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2020. doi: 10.1007/978-3-030-50995-8_1. URL https://doi.org/10.1007/978-3-030-50995-8_1. (Cited on pages 23, 46 and 64.)

[9] Bernhard K. Aichernig, Edi Muškardin, and Andrea Pferscher. Learning-based fuzzing of IoT message brokers. In *14th IEEE Conference on Software Testing, Verification and Validation, ICST 2021, Porto de Galinhas, Brazil, April 12-16, 2021*, pages 47–58. IEEE, 2021. doi: 10.1109/ICST49551.2021.00017. URL https://doi.org/10.1109/ICST49551.2021.00017. (Cited on pages 9, 24 and 158.)

[10] Bernhard K. Aichernig, Edi Muškardin, and Andrea Pferscher. Active vs. passive: A comparison of automata learning paradigms for network protocols. In Matt Luckcuck and Marie Farrell, editors, *Proceedings Fourth International Workshop on Formal Methods for Autonomous Systems (FMAS2022)*, volume 371 of *EPTCS*, pages 1–19, 2022. doi: 10.4204/EPTCS.371.1. URL https://doi.org/10.4204/EPTCS.371.1. (Cited on pages 9, 24, 53 and 166.)

[11] René Alquézar and Alberto Sanfeliu. An algebraic framework to represent finite state machines in single-layer recurrent neural networks. *Neural Computation*, 7(5):931–949, 1995. doi: 10.1162/neco.1995.7.5.931. URL https://doi.org/10.1162/neco.1995.7.5.931. (Cited on pages 100 and 107.)

[12] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2669–2678. AAAI Press, 2018. doi: 10.1609/AAAI.V32I1.11797. URL https://doi.org/10.1609/aaai.v32i1.11797. (Cited on pages 127, 154 and 162.)

[13] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi: 10.1145/1007352.1007390. URL https://doi.org/10.1145/1007352.1007390. (Cited on page 157.)

[14] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. doi: 10.1016/0890-5401(87)90052-6. URL https://doi.org/10.1016/0890-5401(87)90052-6. (Cited on pages 2, 16, 38, 40, 55, 61, 62, 63, 64, 81 and 157.)

[15] Dana Angluin and Michael Kharitonov. When won't membership queries help? (extended abstract). In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 444–454. ACM, 1991. doi: 10.1145/103418.103420. URL https://doi.org/10.1145/103418.103420. (Cited on page 157.)

[16] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: ordering points to identify the clustering structure. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 49–60. ACM Press, 1999. doi: 10.1145/304182.304187. URL https://doi.org/10.1145/304182.304187. (Cited on page 96.)

[17] Giovanni Bacci, Anna Ingólfsdóttir, Kim G. Larsen, and Raphaël Reynouard. Active learning of Markov Decision Processes using Baum-Welch algorithm. In M. Arif Wani, Ishwar K. Sethi,

Weisong Shi, Guangzhi Qu, Daniela Stan Raicu, and Ruoming Jin, editors, *20th IEEE International Conference on Machine Learning and Applications, ICMLA 2021, Pasadena, CA, USA, December 13-16, 2021*, pages 1203–1208, 2021. doi: 10.1109/ICMLA52953.2021.00195. URL https://doi.org/10.1109/ICMLA52953.2021.00195. (Cited on page 156.)

[18] Claudine Badue, Ranik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M. Paixao, Filipe Mutz, Lucas de Paula Veronese, Thiago Oliveira-Santos, and Alberto F. De Souza. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2020.113816. URL https://www.sciencedirect.com/science/article/pii/S095741742030628X. (Cited on pages 1 and 3.)

[19] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. ISBN 978-0-262-02649-9. (Cited on pages 15, 55, 127 and 129.)

[20] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *CoRR*, abs/2003.05991, 2020. URL https://arxiv.org/abs/2003.05991. (Cited on page 3.)

[21] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.*, 13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077. URL https://doi.org/10.1109/TSMC.1983.6313077. (Cited on page 150.)

[22] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA*, pages 125–126. IEEE Computer Society, 2006. doi: 10.1109/QEST.2006.59. URL https://doi.org/10.1109/QEST.2006.59. (Cited on page 156.)

[23] Mordechai Ben-Ari and Francesco Mondada. *Finite State Machines*, pages 55–61. Springer International Publishing, Cham, 2018. ISBN 978-3-319-62533-1. doi: 10.1007/978-3-319-62533-1_4. URL https://doi.org/10.1007/978-3-319-62533-1_4. (Cited on page 11.)

[24] Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the correspondence between conformance testing and regular inference. In Maura Cerioli, editor, *Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3442 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2005. doi: 10.1007/978-3-540-31984-9_14. URL https://doi.org/10.1007/978-3-540-31984-9_14. (Cited on page 23.)

[25] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL http://arxiv.org/abs/1912.06680. (Cited on pages 1 and 4.)

[26] Matteo Biagiola and Paolo Tonella. Testing of deep reinforcement learning agents with surrogate models. *ACM Transactions on Software Engineering and Methodology*, 33(3):73:1–73:33, 2024. doi: 10.1145/3631970. URL https://doi.org/10.1145/3631970. (Cited on page 163.)

[27] Alan W. Biermann and Jerome A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Computers*, 21(6):592–597, 1972. doi: 10.1109/TC.1972.5009015. URL https://doi.org/10.1109/TC.1972.5009015. (Cited on page 159.)

[28] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. (Cited on pages 1, 3 and 144.)

[29] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN 9780387310732. URL https://www.worldcat.org/oclc/71008143. (Cited on pages 3 and 96.)

[30] Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: - runtime enforcement for reactive systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 533–548. Springer, 2015. URL https://doi.org/10.1007/978-3-662-46681-0_51. (Cited on pages 127 and 162.)

[31] Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Neider, and David R. Piegdon. libalf: The automata learning framework. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 360–364, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14295-6. (Cited on page 52.)

[32] Alexander Bork, Sebastian Junges, Joost-Pieter Katoen, and Tim Quatmann. Verification of indefinite-horizon pomdps. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2020. doi: 10.1007/978-3-030-59152-6_16. URL https://doi.org/10.1007/978-3-030-59152-6_16. (Cited on page 116.)

[33] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *CoRR*, abs/1606.01540, 2016. (Cited on pages 118, 122, 142, 143, 146, 149 and 154.)

[34] Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 11079–11091. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/47e288629a6996a17ce50b90a056a0e1-Paper-Conference.pdf. (Cited on page 94.)

[35] Tamim Burgstaller and Edi Muškardin. Modeling Git with AALpy. https://github.com/DES-Lab/Q-learning-under-Partial-Observability, 2024. Accessed: January 17, 2024. (Cited on pages 6, 9, 53, 157, 158 and 166.)

[36] Paolo Camurati and Paolo Prinetto. Formal verification of hardware correctness: Introduction and survey of current research. *Computer*, 21(7):8–19, 1988. doi: 10.1109/2.65. URL https://doi.org/10.1109/2.65. (Cited on page 2.)

[37] Yihan Cao, Siyu Li, Yixin Liu, Zhiling Yan, Yutong Dai, Philip S. Yu, and Lichao Sun. A comprehensive survey of ai-generated content (AIGC): A history of generative AI from GAN to ChatGPT. *CoRR*, abs/2303.04226, 2023. doi: 10.48550/ARXIV.2303.04226. URL https://doi.org/10.48550/arXiv.2303.04226. (Cited on page 1.)

[38] Steven Carr, Nils Jansen, and Ufuk Topcu. Verifiable RNN-based policies for POMDPs under temporal logic constraints. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4121–4127. ijcai.org, 2020. doi: 10.24963/ijcai.2020/570. URL https://doi.org/10.24963/ijcai.2020/570. (Cited on page 160.)

[39] Steven Carr, Nils Jansen, and Ufuk Topcu. Task-aware verifiable RNN-based policies for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 72:819–847, 2021. doi: 10.1613/jair.1.12963. URL https://doi.org/10.1613/jair.1.12963. (Cited on page 162.)

[40] Steven Carr, Nils Jansen, Sebastian Junges, and Ufuk Topcu. Safe reinforcement learning via shielding under partial observability. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 14748–14756. AAAI Press, 2023. doi: 10.1609/aaai.v37i12.26723. URL https://doi.org/10.1609/aaai.v37i12.26723. (Cited on pages 119 and 162.)

[41] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In Rafael C. Carrasco and Jose Oncina, editors, *Grammatical Inference and Applications*, pages 139–152, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. ISBN 978-3-540-48985-6. (Cited on pages 3, 26, 41, 55, 59, 125, 151, 156 and 159.)

[42] Francisco Casacuberta and Enrique Vidal. Machine translation with inferred stochastic finite-state transducers. *Computational Linguistics*, 30(2):205–225, 2004. (Cited on page 156.)

[43] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2*, pages 1023–1028. AAAI Press / The MIT Press, 1994. URL http://www.aaai.org/Library/AAAI/1994/aaai94-157.php. (Cited on pages 116 and 138.)

[44] Davide Castelvecchi. Can we open the black box of AI? *Nature*, 538:20–23, 10 2016. doi: 10.1038/538020a. (Cited on page 4.)

[45] Krishnendu Chatterjee, Martin Chmelik, Raghav Gupta, and Ayush Kanodia. Qualitative analysis of POMDPs with temporal logic specifications for robotics applications. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 325–330. IEEE, 2015. doi: 10.1109/ICRA.2015.7139019. URL https://doi.org/10.1109/ICRA.2015.7139019. (Cited on page 115.)

[46] Chia Yuan Cho, Domagoj Babic, Eui Chul Richard Shin, and Dawn Song. Inference and analysis of formal models of botnet command and control protocols. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 426–439. ACM, 2010. doi: 10.1145/1866307.1866355. URL https://doi.org/10.1145/1866307.1866355. (Cited on page 158.)

[47] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL http://arxiv.org/abs/1409.1259. (Cited on pages 4, 29, 30 and 100.)

[48] Francois Chollet et al. Keras, 2015. URL https://github.com/fchollet/keras. (Cited on page 33.)

[49] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, 4(3):178–187, 1978. doi: 10.1109/TSE.1978.231496. URL https://doi.org/10.1109/TSE.1978.231496. (Cited on pages 18 and 19.)

[50] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI*, pages 183–188, 1992. (Cited on pages 113 and 161.)

[51] Wenjing Chu, Shuo Chen, and Marcello M. Bonsangue. Learning probabilistic automata using residuals. In Antonio Cerone and Peter Csaba Ölveczky, editors, *Theoretical Aspects of Computing - ICTAC 2021 - 18th International Colloquium, Virtual Event, Nur-Sultan, Kazakhstan, September 8-10, 2021, Proceedings*, volume 12819 of *Lecture Notes in Computer Science*, pages 295–313. Springer, 2021. doi: 10.1007/978-3-030-85315-0_17. URL https://doi.org/10.1007/978-3-030-85315-0_17. (Cited on page 156.)

[52] Alexander Clark. Distributional learning of some context-free languages with a minimally adequate teacher. In José M. Sempere and Pedro García, editors, *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings*, volume 6339 of *Lecture Notes in Computer Science*, pages 24–37. Springer, 2010. doi: 10.1007/978-3-642-15488-1_4. URL https://doi.org/10.1007/978-3-642-15488-1_4. (Cited on page 157.)

[53] Axel Cleeremans, David Servan-Schreiber, and James L. McClelland. Finite state automata and simple recurrent networks. *Neural Computing*, 1(3):372–381, 1989. doi: 10.1162/neco.1989.1.3.372. URL https://doi.org/10.1162/neco.1989.1.3.372. (Cited on page 160.)

[54] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, 2002. doi: 10.1109/34.1000236. URL https://doi.org/10.1109/34.1000236. (Cited on page 96.)

[55] European Commision. European Unions AI Act, 2021. URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A52021PC0206. Accessed: January 21, 2024. (Cited on page 1.)

[56] Weidong Cui, Jayanthkumar Kannan, and Helen J. Wang. Discoverer: Automatic protocol reverse engineering from network traces. In Niels Provos, editor, *Proceedings of the 16th USENIX Security Symposium, Boston, MA, USA, August 6-10, 2007*. USENIX Association, 2007. URL https://www.usenix.org/conference/16th-usenix-security-symposium/discoverer-automatic-protocol-reverse-engineering-network. (Cited on page 24.)

[57] Leon Danon, Jordi Duch, Albert Diaz-Guilera, and Alex Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005, 06 2005. doi: 10.1088/1742-5468/2005/09/P09008. (Cited on page 99.)

[58] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, USA, 2010. ISBN 0521763169. (Cited on pages 2, 24, 25 and 41.)

[59] André de Matos Pedro, Paul Andrew Crocker, and Simão Melo de Sousa. Learning stochastic timed automata from sample executions. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part I*, volume 7609 of *Lecture Notes in Computer Science*, pages 508–523. Springer, 2012. doi: 10.1007/978-3-642-34026-0_38. URL https://doi.org/10.1007/978-3-642-34026-0_38. (Cited on page 157.)

[60] Simon Dierl, Falk Maria Howar, Sean Kauffman, Martin Kristjansen, Kim Guldstrand Larsen, Florian Lorber, and Malte Mauritz. Learning symbolic timed models from concrete timed data. In Kristin Yvonne Rozier and Swarat Chaudhuri, editors, *NASA Formal Methods - 15th International Symposium, NFM 2023, Houston, TX, USA, May 16-18, 2023, Proceedings*, volume 13903 of *Lecture Notes in Computer Science*, pages 104–121. Springer, 2023. doi: 10.1007/978-3-031-33170-1_7. URL https://doi.org/10.1007/978-3-031-33170-1_7. (Cited on page 157.)

[61] Simon Dierl, Paul Fiterau-Brostean, Falk Howar, Bengt Jonsson, Konstantinos Sagonas, and Fredrik Tåquist. Scalable tree-based register automata learning. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II*, volume 14571 of *Lecture Notes in Computer Science*, pages 87–108. Springer, 2024. doi: 10.1007/978-3-031-57249-4_5. URL https://doi.org/10.1007/978-3-031-57249-4_5. (Cited on page 157.)

[62] Steven Dilsizian. Artificial intelligence in medicine and cardiac imaging: Harnessing big data and advanced computing to provide personalized medical diagnosis and treatment. *Current cardiology reports*, 16:441, 01 2014. doi: 10.1007/s11886-013-0441-8. (Cited on page 1.)

[63] Taylor Dohmen, Noah Topper, George K. Atia, Andre Beckus, Ashutosh Trivedi, and Alvaro Velasquez. Inferring probabilistic reward machines from non-Markovian reward signals for reinforcement learning. In Akshat Kumar, Sylvie Thiébaux, Pradeep Varakantham, and William Yeoh, editors, *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24, 2022*, pages 574–582. AAAI Press, 2022. URL https://ojs.aaai.org/index.php/ICAPS/article/view/19844. (Cited on page 161.)

[64] Guoliang Dong, Jingyi Wang, Jun Sun, Yang Zhang, Xinyu Wang, Ting Dai, Jin Song Dong, and Xingen Wang. Towards interpreting recurrent neural networks through probabilistic abstraction. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*, pages 499–510. IEEE, 2020. doi: 10.1145/3324884.3416592. URL https://doi.org/10.1145/3324884.3416592. (Cited on pages 5, 24, 93, 94, 97, 105, 160 and 161.)

[65] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 194–202. Morgan Kaufmann, San Francisco (CA), 1995. ISBN 978-1-55860-377-6. doi: https://doi.org/10.1016/B978-1-55860-377-6.50032-3. URL https://www.sciencedirect.com/science/article/pii/B9781558603776500323. (Cited on page 90.)

[66] Khaled El-Fakih, Roland Groz, Muhammad Naeem Irfan, and Muzammil Shahbaz. Learning finite state models of observable nondeterministic systems in a testing context. In *ICTSS 2010*, pages 97–102, 2010. (Cited on pages 16, 40, 155 and 156.)

[67] Jeffrey L. Elman. Finding structure in time. *Cogn. Sci.*, 14(2):179–211, 1990. doi: 10.1207/s15516709cog1402_1. URL https://doi.org/10.1207/s15516709cog1402_1. (Cited on pages 4, 28, 29, 77 and 99.)

[68] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press, 1996. URL http://www.aaai.org/Library/KDD/1996/kdd96-037.php. (Cited on page 96.)

[69] Remi Eyraud, Dakotah Lambert, Badr Tahri Joutei, Aidar Gaffarov, Mathias Cabanne, Jeffrey Heinz, and Chihiro Shibata. TAYSIR competition: Transformer+RNN: Algorithms to yield simple and interpretable representations. In Francois Coste, Faissal Ouardi, and Guillaume Rabusseau, editors, *Proceedings of 16th edition of the International Conference on Grammatical Inference*, volume 217 of *Proceedings of Machine Learning Research*, pages 275–290. PMLR,

10–13 Jul 2023. URL https://proceedings.mlr.press/v217/eyraud23a.html. (Cited on pages 6, 8, 77, 88 and 167.)

[70] Absalom E. Ezugwu, Abiodun M. Ikotun, Olaide Nathaniel Oyelade, Laith Mohammad Abualigah, Jeffrey O. Agushaka, Christopher I. Eke, and Andronicus Ayobami Akinyelu. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Eng. Appl. Artif. Intell.*, 110:104743, 2022. doi: 10.1016/J.ENGAPPAI.2022.104743. URL https://doi.org/10.1016/j.engappai.2022.104743. (Cited on page 3.)

[71] Tiago Ferreira, Harrison Brewton, Loris D'Antoni, and Alexandra Silva. Prognosis: closed-box analysis of network protocol implementations. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 762â774, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383837. doi: 10.1145/3452296.3472938. URL https://doi.org/10.1145/3452296.3472938. (Cited on page 157.)

[72] Ronald Aylmer Fisher. *Statistical methods for research workers*. Edinburgh Oliver & Boyd, 1925. (Cited on page 148.)

[73] Paul Fiterau-Brostean and Falk Howar. Learning-based testing the sliding window behavior of TCP implementations. In Laure Petrucci, Cristina Seceleanu, and Ana Cavalcanti, editors, *Critical Systems: Formal Methods and Automated Verification - Joint 22nd International Workshop on Formal Methods for Industrial Critical Systems - and - 17th International Workshop on Automated Verification of Critical Systems, FMICS-AVoCS 2017, Turin, Italy, September 18-20, 2017, Proceedings*, volume 10471 of *Lecture Notes in Computer Science*, pages 185–200. Springer, 2017. doi: 10.1007/978-3-319-67113-0_12. URL https://doi.org/10.1007/978-3-319-67113-0_12. (Cited on page 157.)

[74] Paul Fiterau-Brostean, Ramon Janssen, and Frits W. Vaandrager. Combining model learning and model checking to analyze TCP implementations. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, volume 9780 of *LNCS*, pages 454–471, 2016. (Cited on page 68.)

[75] Michael Foster, Roland Groz, Catherine Oriat, Adenilso da Silva Simão, Germán Vega, and Neil Walkinshaw. Active inference of efsms without reset. In Yi Li and Sofiène Tahar, editors, *Formal Methods and Software Engineering - 24th International Conference on Formal Engineering Methods, ICFEM 2023, Brisbane, QLD, Australia, November 21-24, 2023, Proceedings*, volume 14308 of *Lecture Notes in Computer Science*, pages 29–46. Springer, 2023. doi: 10.1007/978-981-99-7584-6_3. URL https://doi.org/10.1007/978-981-99-7584-6_3. (Cited on page 158.)

[76] Karl Pearson F.R.S. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900. doi: 10.1080/14786440009463897. URL https://doi.org/10.1080/14786440009463897. (Cited on page 59.)

[77] Daniel Furelos-Blanco, Mark Law, Alessandra Russo, Krysia Broda, and Anders Jonsson. Induction of subgoal automata for reinforcement learning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3890–3897. AAAI Press, 2020. doi: 10.1609/AAAI.V34I04.5802. URL https://doi.org/10.1609/aaai.v34i04.5802. (Cited on page 162.)

[78] Pierre Ganty. Learning the state machine behind a modal text editor: The (neo)vim case study. In *30th International Symposium on Model Checking Software (SPIN 2024)*, 2024. (Cited on page 51.)

[79] Maor Gaon and Ronen I. Brafman. Reinforcement learning with non-Markovian rewards. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3980–3987. AAAI Press, 2020. doi: 10.1609/AAAI.V34I04.5814. URL https://doi.org/10.1609/aaai.v34i04.5814. (Cited on pages 121 and 162.)

[80] Bishwamittra Ghosh and Daniel Neider. A formal language approach to explaining RNNs. *CoRR*, abs/2006.07292, 2020. URL https://arxiv.org/abs/2006.07292. (Cited on page 159.)

[81] Georgios Giantamidis and Stavros Tripakis. Learning Moore machines from input-output traces. In John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou, editors, *FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings*, volume 9995 of *Lecture Notes in Computer Science*, pages 291–309, 2016. doi: 10.1007/978-3-319-48989-6_18. URL https://doi.org/10.1007/978-3-319-48989-6_18. (Cited on pages 25 and 41.)

[82] C. Lee Giles, Clifford B. Miller, Dong Chen, Guo-Zheng Sun, Hsing-Hen Chen, and Yee-Chun Lee. Extracting and learning an unknown grammar with recurrent neural networks. In John E. Moody, Stephen Jose Hanson, and Richard Lippmann, editors, *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 317–324. Morgan Kaufmann, 1991. URL https://proceedings.neurips.cc/paper_files/paper/1991/file/15de21c670ae7c3f6f3f1f37029303c9-Paper.pdf. (Cited on page 160.)

[83] E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3): 302–320, 1978. doi: 10.1016/S0019-9958(78)90562-4. URL https://doi.org/10.1016/S0019-9958(78)90562-4. (Cited on page 2.)

[84] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf. (Cited on pages 1 and 3.)

[85] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org. (Cited on page 1.)

[86] Mark W. Goudreau, C. Lee Giles, Srimat T. Chakradhar, and Dong Chen. First-order versus second-order single-layer recurrent neural networks. *IEEE Trans. Neural Networks*, 5(3):511–513, 1994. doi: 10.1109/72.286928. URL https://doi.org/10.1109/72.286928. (Cited on pages 100 and 107.)

[87] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. Learning of event-recording automata. In *FORMATS 2004 and FTRTFT 2004*, volume 3253, pages 379–396. Springer, 2004. doi: 10.1007/978-3-540-30206-3_26. URL https://doi.org/10.1007/978-3-540-30206-3_26. (Cited on page 162.)

[88] Olga Grinchtein, Martin Leucker, and Nir Piterman. Inferring network invariants automatically. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint*

Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings, volume 4130 of *Lecture Notes in Computer Science*, pages 483–497. Springer, 2006. doi: 10.1007/11814771_40. URL https://doi.org/10.1007/11814771_40. (Cited on page 159.)

[89] Roland Groz, Nicolas Brémond, Adenilso da Silva Simão, and Catherine Oriat. *hW*-inference: A heuristic approach to retrieve models through black box testing. *Journal of Systems and Software*, 159, 2020. doi: 10.1016/J.JSS.2019.110426. URL https://doi.org/10.1016/j.jss.2019.110426. (Cited on pages 53, 158 and 169.)

[90] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979. ISSN 00359254. doi: 10.2307/2346830. URL http://dx.doi.org/10.2307/2346830. (Cited on page 3.)

[91] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. Deepsynth: Automata synthesis for automatic task segmentation in deep reinforcement learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 7647–7656. AAAI Press, 2021. doi: 10.1609/AAAI.V35I9.16935. URL https://doi.org/10.1609/aaai.v35i9.16935. (Cited on pages 126 and 161.)

[92] Matthew J. Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable mdps. In *2015 AAAI Fall Symposia, Arlington, Virginia, USA, November 12-14, 2015*, pages 29–37. AAAI Press, 2015. URL http://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11673. (Cited on pages 28, 77, 113, 122 and 123.)

[93] Marijn Heule and Sicco Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 18(4):825–856, 2013. doi: 10.1007/S10664-012-9222-Z. URL https://doi.org/10.1007/s10664-012-9222-z. (Cited on page 159.)

[94] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018. (Cited on page 122.)

[95] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computing*, 9 (8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735. (Cited on pages 4, 29, 30 and 99.)

[96] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. ISSN 01621459. (Cited on page 59.)

[97] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley Longman Publishing Co., Inc., USA, 1969. (Cited on page 99.)

[98] Bo-Jian Hou and Zhi-Hua Zhou. Learning with interpretable structure from gated RNN. *IEEE Trans. Neural Networks Learn. Syst.*, 31(7):2267–2279, 2020. doi: 10.1109/TNNLS.2020.2967051. URL https://doi.org/10.1109/TNNLS.2020.2967051. (Cited on pages 109 and 161.)

[99] Falk Howar. *Active learning of interface programs*. PhD thesis, Dortmund University of Technology, 2012. URL https://hdl.handle.net/2003/29486. (Cited on pages 39, 40, 81, 157 and 159.)

[100] Falk Howar and Bernhard Steffen. Active automata learning as black-box search and lazy partition refinement. In Nils Jansen, Mariëlle Stoelinga, and Petra van den Bos, editors, *A Journey from*

Process Algebra via Timed Automata to Model Learning - Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday, volume 13560 of *Lecture Notes in Computer Science*, pages 321–338. Springer, 2022. doi: 10.1007/978-3-031-15629-8_17. URL https://doi.org/10.1007/978-3-031-15629-8_17. (Cited on page 64.)

[101] Falk Howar, Bernhard Steffen, and Maik Merten. Automata learning with automated alphabet abstraction refinement. In Ranjit Jhala and David A. Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2011. doi: 10.1007/978-3-642-18275-4_19. URL https://doi.org/10.1007/978-3-642-18275-4_19. (Cited on pages 158 and 169.)

[102] Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring canonical register automata. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, volume 7148 of *Lecture Notes in Computer Science*, pages 251–266. Springer, 2012. doi: 10.1007/978-3-642-27940-9_17. URL https://doi.org/10.1007/978-3-642-27940-9_17. (Cited on page 157.)

[103] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Mach. Learn.*, 110(3):457–506, 2021. doi: 10.1007/S10994-021-05946-3. URL https://doi.org/10.1007/s10994-021-05946-3. (Cited on pages 5 and 166.)

[104] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2112–2121. PMLR, 2018. URL http://proceedings.mlr.press/v80/icarte18a.html. (Cited on pages 121, 124 and 125.)

[105] Rodrigo Toro Icarte, Ethan Waldie, Toryn Q. Klassen, Richard Anthony Valenzano, Margarita P. Castro, and Sheila A. McIlraith. Learning reward machines for partially observable reinforcement learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15497–15508, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/532435c44bec236b471a47a88d63513d-Abstract.html. (Cited on page 161.)

[106] Malte Isberner. *Foundations of active automata learning: an algorithmic perspective*. PhD thesis, Technical University Dortmund, Germany, 2015. URL https://hdl.handle.net/2003/34282. (Cited on pages 40, 157 and 170.)

[107] Malte Isberner and Bernhard Steffen. An abstract framework for counterexample analysis in active automata learning. In Alexander Clark, Makoto Kanazawa, and Ryo Yoshinaka, editors, *Proceedings of the 12th International Conference on Grammatical Inference, ICGI 2014, Kyoto, Japan, September 17-19, 2014*, volume 34 of *JMLR Workshop and Conference Proceedings*, pages 79–93. JMLR.org, 2014. URL http://proceedings.mlr.press/v34/isberner14a.html. (Cited on page 22.)

[108] Malte Isberner and Bernhard Steffen. An abstract framework for counterexample analysis in active automata learning. In Alexander Clark, Makoto Kanazawa, and Ryo Yoshinaka, editors, *The 12th International Conference on Grammatical Inference*, volume 34 of *Proceedings*

of Machine Learning Research, pages 79–93, Kyoto, Japan, 17–19 Sep 2014. PMLR. URL https://proceedings.mlr.press/v34/isberner14a.html. (Cited on page 40.)

[109] Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, volume 8734 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 2014. doi: 10.1007/978-3-319-11164-3_26. URL https://doi.org/10.1007/978-3-319-11164-3_26. (Cited on pages 2, 23, 40, 45, 55, 64, 81, 157, 158 and 170.)

[110] Malte Isberner, Falk Howar, and Bernhard Steffen. Learning register automata: from languages to program structures. *Machine Learning*, 96(1-2):65–98, 2014. doi: 10.1007/S10994-013-5419-7. URL https://doi.org/10.1007/s10994-013-5419-7. (Cited on pages 2, 53 and 157.)

[111] Malte Isberner, Falk Howar, and Bernhard Steffen. The Open-Source LearnLib - A framework for active automata learning. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 487–495. Springer, 2015. doi: 10.1007/978-3-319-21690-4_32. URL https://doi.org/10.1007/978-3-319-21690-4_32. (Cited on pages 19, 20, 33, 36, 40, 52, 55, 157 and 166.)

[112] Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. Safe reinforcement learning using probabilistic shields (invited paper). In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 3:1–3:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. URL https://doi.org/10.4230/LIPIcs.CONCUR.2020.3. (Cited on pages 127 and 162.)

[113] Walter Johnson, James Porter, Stephanie Ackley, and Douglas Ross. Automatic generation of efficient lexical processors using finite state techniques. *Commun. ACM*, 11:805–813, 12 1968. doi: 10.1145/364175.364185. (Cited on page 11.)

[114] Sebastian Junges and Jurriaan Rot. Learning language intersections. In Nils Jansen, Mariëlle Stoelinga, and Petra van den Bos, editors, *A Journey from Process Algebra via Timed Automata to Model Learning - Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday*, volume 13560 of *Lecture Notes in Computer Science*, pages 371–381. Springer, 2022. doi: 10.1007/978-3-031-15629-8_20. URL https://doi.org/10.1007/978-3-031-15629-8_20. (Cited on pages 51, 53 and 166.)

[115] Akshaya Karthikeyan and U Priyakumar. Artificial intelligence: machine learning for chemical sciences. *Journal of Chemical Sciences*, 134, 03 2022. doi: 10.1007/s12039-021-01995-2. (Cited on pages 1 and 3.)

[116] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994. (Cited on pages 38, 40, 55 and 81.)

[117] Igor Khmelnitsky, Daniel Neider, Rajarshi Roy, Xuan Xie, Benoît Barbot, Benedikt Bollig, Alain Finkel, Serge Haddad, Martin Leucker, and Lina Ye. Property-directed verification and robustness certification of recurrent neural networks. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, volume 12971 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2021. doi: 10.1007/978-3-030-88885-5_24. URL https://doi.org/10.1007/978-3-030-88885-5_24. (Cited on pages 159 and 167.)

[118] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980. (Cited on page 100.)

[119] John F. Kolen. Fool's gold: Extracting finite state machines from recurrent network dynamics. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993]*, pages 501–508. Morgan Kaufmann, 1993. (Cited on pages 94 and 160.)

[120] Bettina Könighofer, Florian Lorber, Nils Jansen, and Roderick Bloem. Shield synthesis for reinforcement learning. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part I*, volume 12476 of *Lecture Notes in Computer Science*, pages 290–306. Springer, 2020. URL https://doi.org/10.1007/978-3-030-61362-4_16. (Cited on page 127.)

[121] Bettina Könighofer, Julian Rudolf, Alexander Palmisano, Martin Tappler, and Roderick Bloem. Online shielding for stochastic systems. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods - 13th International Symposium, NFM 2021, Virtual Event, May 24-28, 2021, Proceedings*, volume 12673 of *Lecture Notes in Computer Science*, pages 231–248. Springer, 2021. URL https://doi.org/10.1007/978-3-030-76384-8_15. (Cited on page 162.)

[122] Anurag Koul, Alan Fern, and Sam Greydanus. Learning finite state representations of recurrent policy networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=S1gOpsCctm. (Cited on pages 82, 83, 94, 159 and 167.)

[123] Ramnandan Krishnamurthy, Aravind S. Lakshminarayanan, Peeyush Kumar, and Balaraman Ravindran. Hierarchical reinforcement learning using spatio-temporal abstractions and deep neural networks. *CoRR*, abs/1605.05359, 2016. URL http://arxiv.org/abs/1605.05359. (Cited on page 163.)

[124] David Kubon, Frantisek Mráz, and Ivan Rychtera. Learning automata using dimensional reduction. In *IBERAMIA 2022*, volume 13788, pages 41–52. Springer, 2022. doi: 10.1007/978-3-031-22419-5_4. URL https://doi.org/10.1007/978-3-031-22419-5_4. (Cited on page 162.)

[125] Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Minimization, learning, and conformance testing of boolean programs. In Christel Baier and Holger Hermanns, editors, *CONCUR 2006 – Concurrency Theory*, pages 203–217, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-37377-3. (Cited on page 157.)

[126] Markus A. Kuppe. Teaching TLA+ to engineers at microsoft. In Catherine Dubois and Pierluigi San Pietro, editors, *Formal Methods Teaching: 5th International Workshop, FMTea 2023, Lübeck, Germany, March 6, 2023, Proceedings*, volume 13962 of *Lecture Notes in Computer Science*, pages 66–81. Springer Nature Switzerland, 2022. doi: 10.1007/978-3-031-27534-0_5. URL https://doi.org/10.1007/978-3-031-27534-0_5. (Cited on page 2.)

[127] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Analysis of a gossip protocol in PRISM. *SIGMETRICS Performance Evaluation Review*, 36(3):17–22, 2008. doi: 10.1145/1481506.1481511. URL https://doi.org/10.1145/1481506.1481511. (Cited on page 55.)

[128] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV 2011*, volume 6806 of *LNCS*, pages 585–591, 2011. (Cited on pages 35, 45, 145, 149 and 166.)

[129] Leslie Lamport, John Matthews, Mark R. Tuttle, and Yuan Yu. Specifying and verifying systems with TLA+. In Gilles Muller and Eric Jul, editors, *Proceedings of the 10th ACM SIGOPS European Workshop, Saint-Emilion, France, July 1, 2002*, pages 45–48. ACM, 2002. doi: 10.1145/1133373.1133382. URL https://doi.org/10.1145/1133373.1133382. (Cited on page 2.)

[130] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In Vasant G. Honavar and Giora Slutzki, editors, *Grammatical Inference, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998. doi: 10.1007/BFB0054059. URL https://doi.org/10.1007/BFb0054059. (Cited on pages 43 and 53.)

[131] Kim G. Larsen and Axel Legay. Statistical model checking: Past, present, and future. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, volume 9952 of *Lecture Notes in Computer Science*, pages 3–15, 2016. doi: 10.1007/978-3-319-47166-2_1. URL https://doi.org/10.1007/978-3-319-47166-2_1. (Cited on pages 45, 68 and 70.)

[132] Xiaoxuan Liu, Livia Faes, Aditya Kale, Siegfried Wagner, Dun Fu, Alice Bruynseels, Thushika Mahendiran, Gabriella Moraes, Mohith Shamdas, Christoph Kern, Joseph Ledsam, MD Schmid, Konstantinos Balaskas, Eric Topol, Lucas Bachmann, Pearse Keane, and Alastair Denniston. A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis. *The Lancet Digital Health*, 1, 09 2019. doi: 10.1016/S2589-7500(19)30123-2. (Cited on page 1.)

[133] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Lin Zhao, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. Summary of ChatGPT-related research and perspective towards the future of large language models. *Meta-Radiology*, 1(2):100017, September 2023. ISSN 2950-1628. doi: 10.1016/j.metrad.2023.100017. URL http://dx.doi.org/10.1016/j.metrad.2023.100017. (Cited on pages 1 and 3.)

[134] Lennart Ljung. *System Identification: Theory for the User, PTR Prentice Hall Information and System Sciences Series*. Prentice Hall, New Jersey, 1999. (Cited on page 163.)

[135] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–136, 1982. doi: 10.1109/TIT.1982.1056489. URL https://doi.org/10.1109/TIT.1982.1056489. (Cited on page 144.)

[136] Yuteng Lu, Weidi Sun, and Meng Sun. Towards mutation testing of reinforcement learning systems. *J. Syst. Archit.*, 131:102701, 2022. doi: 10.1016/j.sysarc.2022.102701. URL https://doi.org/10.1016/j.sysarc.2022.102701. (Cited on page 163.)

[137] Chen Lv, Yahui Liu, Xiaosong Hu, Hongyan Guo, Dongpu Cao, and Fei-Yue Wang. Simultaneous observation of hybrid states for cyber-physical systems: A case study of electric vehicle powertrain. *IEEE Trans. Cybern.*, 48(8):2357–2367, 2018. doi: 10.1109/TCYB.2017.2738003. URL https://doi.org/10.1109/TCYB.2017.2738003. (Cited on page 163.)

[138] Marlos C. Machado, Marc G. Bellemare, and Michael H. Bowling. A Laplacian framework for option discovery in reinforcement learning. In *ICML 2017*, volume 70, pages 2295–2304. PMLR, 2017. URL http://proceedings.mlr.press/v70/machado17a.html. (Cited on page 163.)

[139] Andrzej Mackiewicz and Waldemar Ratajczak. Principal components analysis (PCA). *Computers & Geosciences*, 19(3):303–342, 1993. (Cited on page 144.)

[140] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967. (Cited on page 96.)

[141] T. Soni Madhulatha. An overview on clustering methods. *CoRR*, abs/1205.1117, 2012. URL http://arxiv.org/abs/1205.1117. (Cited on page 96.)

[142] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *ICML 2004*, volume 69. ACM, 2004. doi: 10.1145/1015330.1015355. URL https://doi.org/10.1145/1015330.1015355. (Cited on page 163.)

[143] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim Guldstrand Larsen, and Brian Nielsen. Learning Markov decision processes for model checking. In *Proceedings Quantities in Formal Methods, QFM 2012, Paris, France, 28 August 2012.*, pages 49–63, 2012. (Cited on pages 68, 74 and 156.)

[144] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. Learning deterministic probabilistic automata from a model checking perspective. *Mach. Learn.*, 105(2):255–299, 2016. doi: 10.1007/S10994-016-5565-9. URL https://doi.org/10.1007/s10994-016-5565-9. (Cited on pages 3, 26, 41, 42, 55, 59, 68, 74, 113, 122, 123, 138, 156, 159 and 166.)

[145] Franz Mayr and Sergio Yovine. Regular inference on artificial neural networks. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar R. Weippl, editors, *Machine Learning and Knowledge Extraction - Second IFIP TC 5, TC 8/WG 8.4, 8.9, TC 12/WG 12.9 International Cross-Domain Conference, CD-MAKE 2018, Hamburg, Germany, August 27-30, 2018, Proceedings*, volume 11015 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2018. doi: 10.1007/978-3-319-99740-7_25. URL https://doi.org/10.1007/978-3-319-99740-7_25. (Cited on pages 5, 77, 78, 80, 81, 82, 83, 84, 86, 159 and 167.)

[146] Franz Mayr, Sergio Yovine, Matías Carrasco, Federico Pan, and Federico Vilensky. A congruence-based approach to active automata learning from neural language models. In François Coste, Faissal Ouardi, and Guillaume Rabusseau, editors, *International Conference on Grammatical Inference, ICGI 2023, 10-13 July 2023, Rabat, Morocco*, volume 217 of *Proceedings of Machine Learning Research*, pages 250–264. PMLR, 2023. URL https://proceedings.mlr.press/v217/mayr23a.html. (Cited on page 159.)

[147] Q. Mazouni, Helge Spieker, Arnaud Gotlieb, and Mathieu Acher. A review of validation and verification of neural network-based policies for sequential decision making. *CoRR*, abs/2312.09680, 2023. doi: 10.48550/ARXIV.2312.09680. URL https://doi.org/10.48550/arXiv.2312.09680. (Cited on page 1.)

[148] Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In *ICML*, pages 190–196, 1993. (Cited on pages 113 and 161.)

[149] Ramy Medhat, S. Ramesh, Borzoo Bonakdarpour, and Sebastian Fischmeister. A framework for mining hybrid automata from input/output traces. In *EMSOFT 2015*, pages 177–186. IEEE, 2015. doi: 10.1109/EMSOFT.2015.7318273. URL https://doi.org/10.1109/EMSOFT.2015.7318273. (Cited on page 162.)

[150] Karl Meinke. Learning-based testing of cyber-physical systems-of-systems: A platooning study. In *EPEW 2017*, volume 10497, pages 135–151. Springer, 2017. doi: 10.1007/978-3-319-66583-2_9. URL https://doi.org/10.1007/978-3-319-66583-2_9. (Cited on page 163.)

[151] William Merrill and Nikolaos Tsilivis. Extracting finite automata from RNNs using state merging. *CoRR*, abs/2201.12451, 2022. URL https://arxiv.org/abs/2201.12451. (Cited on page 99.)

[152] Joshua J. Michalenko, Ameesh Shah, Abhinav Verma, Richard G. Baraniuk, Swarat Chaudhuri, and Ankit B. Patel. Representing formal languages: A comparison between finite automata and recurrent neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=H1zeHnA9KX. (Cited on pages 97, 99 and 161.)

[153] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., USA, 1967. ISBN 0131655639. (Cited on pages 82, 100 and 107.)

[154] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015. doi: 10.1038/nature14236. (Cited on page 32.)

[155] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/mniha16.html. (Cited on pages 113, 123, 139 and 147.)

[156] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X. (Cited on pages 44, 81, 86 and 159.)

[157] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956. (Cited on page 2.)

[158] Edi Muškardi and Stefan Pranger. Implementation of Automata Learning Meets Shielding. https://github.com/DES-Lab/Automata-Learning-meets-Shielding, 2022. Accessed: January 24, 2024. (Cited on page 7.)

[159] Edi Muškardin and Martin Tappler. Implementation of CAS-TLE and experimental setup. https://github.com/DES-Lab/Learning-Environment-Models-with-Continuous-Stochastic-Dynamics, 2022. Accessed: January 17, 2024. (Cited on pages 7 and 149.)

[160] Edi Muškardin and Martin Tappler. Implementation and experimental setup for "Reinforcement learning under partial observability guided by learned environment models". https://github.

com/DES-Lab/Q-learning-under-Partial-Observability, 2022. Accessed: January 17, 2024. (Cited on page 7.)

[161] Edi Muškardin and Martin Tappler. Implementation and experimental setup for "Learning Finite State Models from Recurrent Neural Networks". https://github.com/DES-Lab/Extracting-FSM-From-RNNs, 2022. Accessed: January 17, 2024. (Cited on page 7.)

[162] Edi Muškardin and Martin Tappler. Implementation and experimental setup for "Analysis of Clustering-Based Abstraction of RNN State Vectors". https://github.com/DES-Lab/Clustering_RNN_hidden_state_space, 2023. Accessed: January 17, 2024. (Cited on page 7.)

[163] Edi Muškardin and Martin Tappler. Implementation of model extraction method used for the TAYSIR competition. https://github.com/emuskardin/taysir_competition_mbt, 2023. Accessed: January 17, 2024. (Cited on pages 7 and 88.)

[164] Edi Muškardin, Bernhard K. Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. AALpy: An active automata learning library. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, volume 12971 of *Lecture Notes in Computer Science*, pages 67–73. Springer, 2021. doi: 10.1007/978-3-030-88885-5_5. URL https://doi.org/10.1007/978-3-030-88885-5_5. (Cited on pages 6, 7, 8, 33 and 155.)

[165] Edi Muškardin, Ingo Pill, Martin Tappler, and Bernhard K. Aichernig. Automata learning enabling model-based diagnosis. In *32nd International Workshop on Principle of Diagnosis, Hamburg-Germany, September 13th-15th*, 2021. (Cited on pages 9, 53 and 166.)

[166] Edi Muškardin, Bernhard K. Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. AALpy: an active automata learning library. *Innovations in Systems and Software Engineering*, 18(3): 417–426, 2022. doi: 10.1007/S11334-022-00449-3. URL https://doi.org/10.1007/s11334-022-00449-3. (Cited on pages 6, 8, 19, 33, 99, 122, 132, 149 and 155.)

[167] Edi Muškardin, Bernhard K. Aichernig, Ingo Pill, and Martin Tappler. Learning finite state models from recurrent neural networks. In Maurice H. ter Beek and Rosemary Monahan, editors, *Integrated Formal Methods - 17th International Conference, IFM 2022, Lugano, Switzerland, June 7-10, 2022, Proceedings*, volume 13274 of *Lecture Notes in Computer Science*, pages 229–248. Springer, 2022. doi: 10.1007/978-3-031-07727-2_13. URL https://doi.org/10.1007/978-3-031-07727-2_13. (Cited on pages 6, 7, 20, 77 and 155.)

[168] Edi Muškardin, Martin Tappler, and Bernhard K. Aichernig. Testing-based black-box extraction of simple models from RNNs and transformers. In François Coste, Faissal Ouardi, and Guillaume Rabusseau, editors, *International Conference on Grammatical Inference, ICGI 2023, 10-13 July 2023, Rabat, Morocco*, volume 217 of *Proceedings of Machine Learning Research*, pages 291–294. PMLR, 2023. URL https://proceedings.mlr.press/v217/muskardin23a.html. (Cited on pages 6, 8, 77 and 155.)

[169] Edi Muškardin, Martin Tappler, Bernhard K. Aichernig, and Ingo Pill. Reinforcement learning under partial observability guided by learned environment models. In Paula Herber and Anton Wijs, editors, *iFM 2023 - 18th International Conference, iFM 2023, Leiden, The Netherlands, November 13-15, 2023, Proceedings*, volume 14300 of *Lecture Notes in Computer Science*, pages 257–276. Springer, 2023. doi: 10.1007/978-3-031-47705-8_14. URL https://doi.org/10.1007/978-3-031-47705-8_14. (Cited on pages 6, 8, 113 and 155.)

[170] Edi Muškardin, Tamim Burgstaller, Martin Tapper, and Bernhard K. Aichernig. Active model learning of Git version control system. In *20th Workshop on Advances in Model Based Testing (A-MOST 2024)*, 2024. (Cited on page 52.)

[171] Edi Muškardin, Martin Tappler, Bernhard K. Aichernig, and Ingo Pill. Active model learning of stochastic reactive systems (extended version). *Software and System Modelling*, 23(2): 503–524, 2024. doi: 10.1007/S10270-024-01158-0. URL https://doi.org/10.1007/s10270-024-01158-0. (Cited on pages 6, 8, 55 and 155.)

[172] Edi Muškardin, Martin Tappler, Ingo Pill, Bernhard Aichernig, and Thomas Pock. On the relationship between RNN hidden-state vectors and semantic structures. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics ACL 2024*, pages 5641–5658, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.335. (Cited on pages 6, 8 and 93.)

[173] Edi Muškardin et al. Implementation of AALpy: an active automata learning library. https://github.com/DES-Lab/AALpy, 2021. Accessed: January 24, 2024. (Cited on page 7.)

[174] Meike Nauta, Jan Trienes, Shreyasi Pathak, Elisa Nguyen, Michelle Peters, Yasmin Schmitt, Jörg Schlötterer, Maurice van Keulen, and Christin Seifert. From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable ai. *ACM Comput. Surv.*, 55 (13s), jul 2023. ISSN 0360-0300. doi: 10.1145/3583558. URL https://doi.org/10.1145/3583558. (Cited on page 4.)

[175] Tertulien Ndjountche. *Digital Electronics 3: Finite-state Machines*. 10 2016. ISBN 9781848219861. doi: 10.1002/9781119371083. (Cited on page 11.)

[176] Daniel Neider. Learning visibly one-counter automata in polynomial time. 2010. URL https://api.semanticscholar.org/CorpusID:10896704. (Cited on page 157.)

[177] Daniel Neider, Rick Smetsers, Frits W. Vaandrager, and Harco Kuppens. Benchmarks for automata learning and conformance testing. In Tiziana Margaria, Susanne Graf, and Kim G. Larsen, editors, *Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, volume 11200 of *Lecture Notes in Computer Science*, pages 390–416. Springer, 2018. doi: 10.1007/978-3-030-22348-9_23. URL https://doi.org/10.1007/978-3-030-22348-9_23. (Cited on pages 35, 45, 48 and 49.)

[178] Daniel Neider, Jean-Raphaël Gaglione, Ivan Gavran, Ufuk Topcu, Bo Wu, and Zhe Xu. Advice-guided reinforcement learning in a non-Markovian environment. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 9073–9080. AAAI Press, 2021. doi: 10.1609/AAAI.V35I10.17096. URL https://doi.org/10.1609/aaai.v35i10.17096. (Cited on pages 121 and 161.)

[179] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958. ISSN 0002-9939, 1088-6826/e. (Cited on page 57.)

[180] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. DyNet: The dynamic neural network toolkit. *CoRR*, abs/1701.03980, 2017. URL http://arxiv.org/abs/1701.03980. (Cited on page 83.)

[181] Chris Newcombe. Why Amazon chose TLA +. In Yamine Aït Ameur and Klaus-Dieter Schewe, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings*, volume 8477 of *Lecture Notes in Computer*

Science, pages 25–39. Springer, 2014. doi: 10.1007/978-3-662-43652-3_3. URL https://doi.org/10.1007/978-3-662-43652-3_3. (Cited on page 2.)

[182] Oliver Niggemann, Benno Stein, Asmir Vodencarevic, Alexander Maier, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. In *AAAI 2012*. AAAI Press, 2012. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4993. (Cited on page 162.)

[183] Masashi Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 10(1):29–35, 1959. ISSN 1572-9052. doi: 10.1007/BF02883985. URL http://dx.doi.org/10.1007/BF02883985. (Cited on page 68.)

[184] Christian Oliva and Luis F. Lago-Fernández. On the interpretation of recurrent neural networks as finite state machines. In Igor V. Tetko, Vera Kurková, Pavel Karpov, and Fabian J. Theis, editors, *Artificial Neural Networks and Machine Learning - ICANN 2019: Theoretical Neural Computation - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part I*, volume 11727 of *Lecture Notes in Computer Science*, pages 312–323. Springer, 2019. doi: 10.1007/978-3-030-30487-4_25. URL https://doi.org/10.1007/978-3-030-30487-4_25. (Cited on page 82.)

[185] Christian Oliva and Luis Fernando Lago-Fernandez. Stability of internal states in recurrent neural networks trained on regular languages. *Neurocomputing*, 452:212–223, 2021. doi: 10.1016/j.neucom.2021.04.058. URL https://doi.org/10.1016/j.neucom.2021.04.058. (Cited on pages 77 and 78.)

[186] Christian W. Omlin and C. Lee Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996. doi: 10.1016/0893-6080(95)00086-0. URL https://doi.org/10.1016/0893-6080(95)00086-0. (Cited on pages 77 and 160.)

[187] Jose Oncina and Pedro Garcia. Inferring regular languages in polynomial update time. *World Scientific*, 01 1992. doi: 10.1142/9789812797902_0004. (Cited on pages 3, 25, 41, 126 and 159.)

[188] OpenAI. ChatGPT. https://chat.openai.com, 2023. (Cited on page 1.)

[189] Hae-Won Park, Alireza Ramezani, and J.W. Grizzle. A finite-state machine for accommodating unexpected large ground-height variations in bipedal robot walking. *Robotics, IEEE Transactions on*, 29:331–345, 04 2013. doi: 10.1109/TRO.2012.2230992. (Cited on page 11.)

[190] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. (Cited on pages 29, 33 and 166.)

[191] 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*, 12:2825–2830, 2011. (Cited on pages 96 and 166.)

[192] Doron A. Peled, Moshe Y. Vardi, and Mihalis Yannakakis. Black box checking. In Jianping Wu, Samuel T. Chanson, and Qiang Gao, editors, *Formal Methods for Protocol Engineering and Distributed Systems, FORTE XII / PSTV XIX'99, IFIP TC6 WG6.1 Joint International Conference on*

Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX), October 5-8, 1999, Beijing, China, volume 156 of *IFIP Conference Proceedings*, pages 225–240. Kluwer, 1999. (Cited on page 2.)

[193] Josep Perlas, Dir Beltran, and Josefina Rosich. *DESIGN AND IMPLEMENTATION OF A FINITE STATE MACHINE QUEUING TOOL FOR EPICS*. 01 1999. (Cited on page 11.)

[194] Andrea Pferscher. *Automata Learning for Security Testing and Analysis in Networked Environments*. PhD thesis, Graz University of Technology, Austria, 2023. URL https://apferscher.github.io/docs/phd-thesis.pdf. (Cited on page 51.)

[195] Andrea Pferscher and Bernhard K. Aichernig. Learning abstracted non-deterministic finite state machines. In *Testing Software and Systems - 32nd IFIP WG 6.1 International Conference, ICTSS 2020, Naples, Italy, December 9-11, 2020, Proceedings*, pages 52–69, 2020. doi: 10.1007/978-3-030-64881-7_4. URL https://doi.org/10.1007/978-3-030-64881-7_4. (Cited on pages 40 and 156.)

[196] Andrea Pferscher and Bernhard K. Aichernig. Fingerprinting bluetooth low energy devices via active automata learning. In Marieke Huisman, Corina Păsăreanu, and Naijun Zhan, editors, *Formal Methods*, pages 524–542, Cham, 2021. Springer International Publishing. ISBN 978-3-030-90870-6. (Cited on pages 51, 53, 68, 158 and 166.)

[197] Andrea Pferscher and Bernhard K. Aichernig. Stateful black-box fuzzing of bluetooth devices using automata learning. In Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez, editors, *NASA Formal Methods*, pages 373–392, Cham, 2022. Springer International Publishing. ISBN 978-3-031-06773-0. (Cited on pages 52, 53, 158 and 166.)

[198] Andrea Pferscher, Benjamin Wunderling, Bernhard K. Aichernig, and Edi Muškardin. Mining digital twins of a VPN server. In Stefan Hallerstede and Eduard Kamburjan, editors, *Proceedings of the Workshop on Applications of Formal Methods and Digital Twins co-located with 25th International Symposium on Formal Methods (FM 2023), Lübeck, Germany, March 06, 2023*, volume 3507 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL https://ceur-ws.org/Vol-3507/paper6.pdf. (Cited on pages 6, 9, 11, 53, 157, 158 and 166.)

[199] Stefan Pranger, Bettina Könighofer, Lukas Posch, and Roderick Bloem. TEMPEST - synthesis tool for reactive systems and shields in probabilistic environments. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, volume 12971 of *Lecture Notes in Computer Science*, pages 222–228. Springer, 2021. URL https://doi.org/10.1007/978-3-030-88885-5_15. (Cited on pages 131 and 132.)

[200] Stefan Pranger, Bettina Könighofer, Martin Tappler, Martin Deixelberger, Nils Jansen, and Roderick Bloem. Adaptive shielding under uncertainty. In *2021 American Control Conference, ACC 2021, New Orleans, LA, USA, May 25-28, 2021*, pages 3467–3474. IEEE, 2021. doi: 10.23919/ACC50511.2021.9482889. URL https://doi.org/10.23919/ACC50511.2021.9482889. (Cited on page 162.)

[201] Raffaele Pugliese, Stefano Regondi, and Riccardo Marini. Machine learning-based approach: global trends, research directions, and regulatory standpoints. *Data Science and Management*, 4: 19–29, 2021. ISSN 2666-7649. doi: https://doi.org/10.1016/j.dsm.2021.12.002. URL https://www.sciencedirect.com/science/article/pii/S2666764921000485. (Cited on page 1.)

[202] Antonin Raffin. stable-baselines3 A2C agent for lunar lander. https://huggingface.co/sb3/a2c-LunarLander-v2, Jun 2022. Accessed: January 17, 2024. (Cited on page 150.)

[203] Antonin Raffin. stable-baselines3 A2C agent for mountain car. https://huggingface.co/sb3/a2c-MountainCar-v0, Jun 2022. Accessed: January 17, 2024. (Cited on page 150.)

[204] Antonin Raffin. stable-baselines3 DQN agent for cartpole. https://huggingface.co/sb3/dqn-CartPole-v1, Jun 2022. Accessed: January 17, 2024. (Cited on page 152.)

[205] Antonin Raffin. stable-baselines3 DQN agent for lunar lander. https://huggingface.co/sb3/dqn-LunarLander-v2, Jun 2022. Accessed: January 17, 2024. (Cited on page 150.)

[206] Antonin Raffin. stable-baselines3 DQN agent for lunar lander (high reward). https://huggingface.co/araffin/dqn-LunarLander-v2, May 2022. Accessed: January 17, 2024. (Cited on page 152.)

[207] Antonin Raffin. stable-baselines3 PPO agent for cartpole. https://huggingface.co/sb3/ppo-CartPole-v1, Jun 2022. Accessed: January 17, 2024. (Cited on page 152.)

[208] Antonin Raffin. stable-baselines3 PPO agent for lunar lander. https://huggingface.co/sb3/ppo-LunarLander-v2, Jun 2022. Accessed: January 17, 2024. (Cited on page 150.)

[209] Antonin Raffin. stable-baselines3 PPO agent for lunar lander (high reward). https://huggingface.co/araffin/ppo-LunarLander-v2, Jun 2022. Accessed: January 17, 2024. (Cited on page 152.)

[210] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html. (Cited on page 149.)

[211] Partha Pratim Ray. ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, 3:121–154, 2023. ISSN 2667-3452. doi: https://doi.org/10.1016/j.iotcps.2023.04.003. URL https://www.sciencedirect.com/science/article/pii/S266734522300024X. (Cited on pages 1 and 3.)

[212] Raphaël Reynouard. *On learning stochastic models: from theory to practice*. PhD thesis, ReykjavÄ±k University, Iceland, 2023. (Cited on page 53.)

[213] Raphaël Reynouard, Anna Ingólfsdóttir, and Giovanni Bacci. Jajapy: A learning library for stochastic models. In Nils Jansen and Mirco Tribastone, editors, *Quantitative Evaluation of Systems*, pages 30–46, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-43835-6. (Cited on pages 53 and 156.)

[214] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In John DeNero, Mark Finlayson, and Sravana Reddy, editors, *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-3020. URL https://aclanthology.org/N16-3020. (Cited on page 1.)

[215] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993. (Cited on pages 2, 20, 22, 64 and 158.)

[216] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016. (Cited on pages 30 and 31.)

[217] Scapy. Scapy, 2021. URL https://github.com/secdev/scapy/. accessed: October 8, 2021. (Cited on pages 33, 51 and 166.)

[218] Ingo Schellhammer, Joachim Diederich, Michael Towsey, and Claudia Brugman. Knowledge extraction and recurrent neural networks: An analysis of an Elman network trained on a natural language learning task. In David M. W. Powers, editor, *Proceedings of the Joint Conference on New Methods in Language Processing and Computational Natural Language Learning, NeMLaP/CoNLL 1998, Macquarie University, Sydney, NSW, Australia, January 11-17, 1998*, pages 73–78. ACL, 1998. URL https://aclanthology.org/W98-1209/. (Cited on page 160.)

[219] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347. (Cited on page 150.)

[220] Muzammil Shahbaz and Roland Groz. Inferring Mealy machines. In Ana Cavalcanti and Dennis Dams, editors, *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2009. doi: 10.1007/978-3-642-05089-3_14. URL https://doi.org/10.1007/978-3-642-05089-3_14. (Cited on pages 20, 22, 41, 62 and 64.)

[221] Ashish Kumar Shakya, Gopinatha Pillai, and Sohom Chakrabarty. Reinforcement learning algorithms: A brief survey. *Expert Syst. Appl.*, 231:120495, 2023. doi: 10.1016/J.ESWA.2023.120495. URL https://doi.org/10.1016/j.eswa.2023.120495. (Cited on page 31.)

[222] Andy Shih, Adnan Darwiche, and Arthur Choi. Verifying binarized neural networks by Angluin-style learning. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 354–370. Springer, 2019. doi: 10.1007/978-3-030-24258-9_25. URL https://doi.org/10.1007/978-3-030-24258-9_25. (Cited on page 160.)

[223] Hava T. Siegelmann and Eduardo D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991. ISSN 0893-9659. doi: https://doi.org/10.1016/0893-9659(91)90080-F. URL https://www.sciencedirect.com/science/article/pii/089396599190080F. (Cited on page 81.)

[224] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/NATURE16961. URL https://doi.org/10.1038/nature16961. (Cited on pages 1 and 4.)

[225] Satinder P. Singh, Tommi S. Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *ICML*, pages 284–292. Morgan Kaufmann, 1994. (Cited on page 113.)

[226] Rick Smetsers, Paul Fiterau-Brostean, and Frits W. Vaandrager. Model learning as a satisfiability modulo theories problem. In Shmuel Tomi Klein, Carlos Martín-Vide, and Dana Shapira, editors, *Language and Automata Theory and Applications - 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings*, volume 10792 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2018. doi: 10.1007/978-3-319-77313-1_14. URL https://doi.org/10.1007/978-3-319-77313-1_14. (Cited on page 159.)

[227] Lindsay I Smith. A tutorial on principal components analysis. Technical report, Cornell University, USA, February 26 2002. URL http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf. (Cited on page 3.)

[228] Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In Marco Bernardo and Valérie Issarny, editors, *Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, volume 6659 of *Lecture Notes in Computer Science*, pages 256–296. Springer, 2011. doi: 10.1007/978-3-642-21455-4_8. URL https://doi.org/10.1007/978-3-642-21455-4_8. (Cited on pages 23 and 39.)

[229] Student. The probable error of a mean. *Biometrika*, 6(1):1–25, 03 1908. ISSN 0006-3444. doi: 10.1093/biomet/6.1.1. URL https://doi.org/10.1093/biomet/6.1.1. (Cited on page 70.)

[230] Marnix Suilen, Thiago D. Simão, David Parker, and Nils Jansen. Robust anytime learning of Markov decision processes. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b931c44c35ce09e942edab7003eb3daa-Abstract-Conference.html. (Cited on page 156.)

[231] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html. (Cited on page 31.)

[232] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1422–1432. The Association for Computational Linguistics, 2015. doi: 10.18653/v1/d15-1167. URL https://doi.org/10.18653/v1/d15-1167. (Cited on pages 28 and 77.)

[233] Martin Tappler. *Learning-Based Testing in Networked Environments in the Presence of Timed and Stochastic Behaviour*. PhD thesis, Graz University of Technology, Austria, 2019. URL https://mtappler.files.wordpress.com/2019/12/thesis.pdf. (Cited on pages 19 and 81.)

[234] Martin Tappler, Bernhard K. Aichernig, and Roderick Bloem. Model-based testing IoT communication via active automata learning. In *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*, pages 276–287. IEEE Computer Society, 2017. doi: 10.1109/ICST.2017.32. URL https://doi.org/10.1109/ICST.2017.32. (Cited on pages 11, 19, 55, 68 and 99.)

[235] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim Guldstrand Larsen. L^*-based learning of Markov decision processes. In *Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Porto, Portugal, October 7-11, 2019, Proceedings*, pages 651–669, 2019. (Cited on pages 55, 56 and 58.)

[236] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. Time to learn - learning timed automata from tests. In Étienne André and Mariëlle Stoelinga, editors, *Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27-29, 2019, Proceedings*, volume 11750 of *Lecture Notes in Computer Science*, pages 216–235. Springer, 2019. doi: 10.1007/978-3-030-29662-9_13. URL https://doi.org/10.1007/978-3-030-29662-9_13. (Cited on pages 2 and 157.)

[237] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. L*-based learning of Markov decision processes (extended version). *FAC*, 2021. (Cited on pages 16, 45, 49, 55, 56, 58, 59, 60, 61, 62, 63, 66, 67, 68, 69, 74, 76, 117, 156 and 166.)

[238] Martin Tappler, Edi Muškardin, Bernhard K. Aichernig, and Ingo Pill. Active model learning of stochastic reactive systems. In Radu Calinescu and Corina S. Pasareanu, editors, *Software Engineering and Formal Methods - 19th International Conference, SEFM 2021, Virtual Event, December 6-10, 2021, Proceedings*, volume 13085 of *Lecture Notes in Computer Science*, pages 481–500. Springer, 2021. doi: 10.1007/978-3-030-92124-8_27. URL https://doi.org/10.1007/978-3-030-92124-8_27. (Cited on pages 6, 7, 8, 55, 155 and 156.)

[239] Martin Tappler, Bernhard K. Aichernig, and Florian Lorber. Timed automata learning via SMT solving. In Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez, editors, *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*, volume 13260 of *Lecture Notes in Computer Science*, pages 489–507. Springer, 2022. doi: 10.1007/978-3-031-06773-0_26. URL https://doi.org/10.1007/978-3-031-06773-0_26. (Cited on pages 157 and 159.)

[240] Martin Tappler, Filip Cano Córdoba, Bernhard K. Aichernig, and Bettina Könighofer. Search-based testing of reinforcement learning. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 503–510. ijcai.org, 2022. doi: 10.24963/ijcai.2022/72. URL https://doi.org/10.24963/ijcai.2022/72. (Cited on page 163.)

[241] Martin Tappler, Stefan Pranger, Bettina Könighofer, Edi Muškardin, Roderick Bloem, and Kim G. Larsen. Automata learning meets shielding. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part I*, volume 13701 of *Lecture Notes in Computer Science*, pages 335–359. Springer, 2022. doi: 10.1007/978-3-031-19849-6_20. URL https://doi.org/10.1007/978-3-031-19849-6_20. (Cited on pages 6, 8, 127 and 155.)

[242] Martin Tappler, Edi Muškardin, Bernhard K. Aichernig, and Bettina Könighofer. Learning Environment Models with Continuous Stochastic Dynamics – with an Application to Deep RL Testing. In *17th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2024. (Cited on pages 6, 7, 8, 141 and 155.)

[243] Martin Tappler, Andrea Pferscher, Bernhard K. Aichernig, and Bettina Könighofer. Learning and repair of deep reinforcement learning policies from fuzz-testing data. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 6:1–6:13. ACM, 2024. doi: 10.1145/3597503.3623311. URL https://doi.org/10.1145/3597503.3623311. (Cited on pages 154 and 163.)

[244] Tiobe Index, October 2018. URL https://www.tiobe.com/tiobe-index/. (Cited on page 33.)

[245] M. Tomita. Dynamic construction of finite automata from examples using hill-climbing. In *Conference of the Cognitive Science Society*, pages 105–108, 1982. (Cited on pages 82 and 99.)

[246] Lisa Torrey and Matthew E. Taylor. Teaching on a budget: agents advising agents in reinforcement learning. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1053–1060. IFAAMAS, 2013. URL http://dl.acm.org/citation.cfm?id=2485086. (Cited on page 147.)

[247] Jan Tretmans. Model based testing with labelled transition systems. In Robert M. Hierons, Jonathan P. Bowen, and Mark Harman, editors, *Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer, 2008. doi: 10.1007/978-3-540-78917-8_1. URL https://doi.org/10.1007/978-3-540-78917-8_1. (Cited on page 18.)

[248] Miller Trujillo, Mario Linares-Vásquez, Camilo Escobar-Velásquez, Ivana Dusparic, and Nicolás Cardozo. Does neuron coverage matter for deep reinforcement learning?: A preliminary study. In *ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*, pages 215–220. ACM, 2020. doi: 10.1145/3387940.3391462. URL https://doi.org/10.1145/3387940.3391462. (Cited on page 163.)

[249] Frits W. Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2022. doi: 10.1007/978-3-030-99524-9_12. URL https://doi.org/10.1007/978-3-030-99524-9_12. (Cited on page 81.)

[250] Frits W. Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2022. doi: 10.1007/978-3-030-99524-9_12. URL https://doi.org/10.1007/978-3-030-99524-9_12. (Cited on page 159.)

[251] Frits W. Vaandrager, Masoud Ebrahimi, and Roderick Bloem. Learning Mealy machines with one timer. *Inf. Comput.*, 295(Part B):105013, 2023. doi: 10.1016/J.IC.2023.105013. URL https://doi.org/10.1016/j.ic.2023.105013. (Cited on page 157.)

[252] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134â1142, nov 1984. ISSN 0001-0782. doi: 10.1145/1968.1972. URL https://doi.org/10.1145/1968.1972. (Cited on pages 18, 44, 81, 157 and 159.)

[253] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html. (Cited on page 3.)

[254] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. (Cited on pages 3 and 4.)

[255] Sicco Verwer and Christian A. Hammerschmidt. flexfringe: A passive automaton learning package. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 638–642, 2017. doi: 10.1109/ICSME.2017.58. (Cited on pages 43 and 52.)

[256] Sicco Verwer and Christian A. Hammerschmidt. flexfringe: Modeling software behavior by learning probabilistic automata. *CoRR*, abs/2203.16331, 2022. doi: 10.48550/ARXIV.2203.16331. URL https://doi.org/10.48550/arXiv.2203.16331. (Cited on page 53.)

[257] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. An algorithm for learning real-time automata. 2007. URL https://api.semanticscholar.org/CorpusID:15561445. (Cited on page 157.)

[258] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In José M. Sempere and Pedro García, editors, *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings*, volume 6339 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 2010. doi: 10.1007/978-3-642-15488-1_17. URL https://doi.org/10.1007/978-3-642-15488-1_17. (Cited on pages 157, 162 and 163.)

[259] Giulia Vilone and Luca Longo. Explainable artificial intelligence: a systematic review. *ArXiv*, abs/2006.00093, 2020. URL https://api.semanticscholar.org/CorpusID:219176929. (Cited on page 1.)

[260] Michele Volpato and Jan Tretmans. Approximate active learning of nondeterministic input output transition systems. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 72, 2015. doi: 10.14279/TUJ.ECEASST.72.1008. URL https://doi.org/10.14279/tuj.eceasst.72.1008. (Cited on page 156.)

[261] Masaki Waga, Ezequiel Castellano, Sasinee Pruekprasert, Stefan Klikovits, Toru Takisaka, and Ichiro Hasuo. Dynamic shielding for reinforcement learning in black-box environments. *CoRR*, abs/2207.13446, 2022. doi: 10.48550/arXiv.2207.13446. URL https://doi.org/10.48550/arXiv.2207.13446. (Cited on page 162.)

[262] Neil Walkinshaw and Kirill Bogdanov. Automated comparison of state-based software models in terms of their language and structure. *ACM Trans. Softw. Eng. Methodol.*, 22(2):13:1–13:37, 2013. doi: 10.1145/2430545.2430549. URL https://doi.org/10.1145/2430545.2430549. (Cited on page 82.)

[263] Neil Walkinshaw, John Derrick, and Qiang Guo. Iterative refinement of reverse-engineered models by model-based testing. In Ana Cavalcanti and Dennis Dams, editors, *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2009. doi: 10.1007/978-3-642-05089-3_20. URL https://doi.org/10.1007/978-3-642-05089-3_20. (Cited on page 42.)

[264] Felix Wallner, Bernhard K. Aichernig, and Christian Burghard. It's not a feature, it's a bug: Fault-tolerant model mining from noisy data. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 29:1–29:13. ACM, 2024. doi: 10.1145/3597503.3623346. URL https://doi.org/10.1145/3597503.3623346. (Cited on page 159.)

[265] Cheng Wang and Mathias Niepert. State-regularized recurrent neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6596–6606. PMLR, 2019. URL http://proceedings.mlr.press/v97/wang19j.html. (Cited on page 82.)

[266] Qinglong Wang, Kaixuan Zhang, Alexander G. Ororbia II, Xinyu Xing, Xue Liu, and C. Lee Giles. An empirical evaluation of rule extraction from recurrent neural networks. *Neural Computing*, 30 (9), 2018. doi: 10.1162/neco_a_01111. URL https://doi.org/10.1162/neco_a_01111. (Cited on pages 160 and 167.)

[267] Qinglong Wang, Kaixuan Zhang, Xue Liu, and C. Lee Giles. Verification of recurrent neural networks through rule extraction. *CoRR*, abs/1811.06029, 2018. URL http://arxiv.org/abs/1811.06029. (Cited on page 159.)

[268] Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1442–1451. The Association for Computational Linguistics, 2016. doi: 10.18653/v1/n16-1170. URL https://doi.org/10.18653/v1/n16-1170. (Cited on pages 28 and 77.)

[269] Christopher J. C. H. Watkins and Peter Dayan. Technical note Q-learning. *Mach. Learn.*, 8: 279–292, 1992. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698. (Cited on pages 4, 32, 113, 119 and 122.)

[270] Raymond L. Watrous and Gary M. Kuhn. Induction of finite-state automata using second-order recurrent networks. In John E. Moody, Stephen Jose Hanson, and Richard Lippmann, editors, *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 309–317. Morgan Kaufmann, 1991. (Cited on page 160.)

[271] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5244–5253. PMLR, 2018. URL http://proceedings.mlr.press/v80/weiss18a.html. (Cited on pages 5, 77, 78, 80, 82, 83, 84, 86, 87, 93, 94, 99, 159 and 167.)

[272] Gail Weiss, Yoav Goldberg, and Eran Yahav. Learning deterministic weighted automata with queries and counterexamples. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8558–8569, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/d3f93e7766e8e1b7ef66dfdd9a8be93b-Abstract.html. (Cited on page 159.)

[273] Yuhuai Wu, Elman Mansimov, Roger B. Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In *NIPS*, 2017. (Cited on page 123.)

[274] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2, 08 2015. doi: 10.1007/s40745-015-0040-1. (Cited on page 96.)

[275] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. Joint inference of reward machines and policies for reinforcement learning. In J. Christopher Beck, Olivier Buffet, Jörg Hoffmann, Erez Karpas, and Shirin Sohrabi, editors, *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 590–598. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/ICAPS/article/view/6756. (Cited on page 161.)

[276] Zhe Xu, Bo Wu, Aditya Ojha, Daniel Neider, and Ufuk Topcu. Active finite reward automaton inference and reinforcement learning using queries and counterexamples. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar R. Weippl, editors, *Machine Learning and Knowledge Extraction - 5th IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2021, Virtual Event, August 17-20, 2021, Proceedings*, volume 12844 of *Lecture Notes in Computer Science*, pages 115–135. Springer, 2021. doi: 10.1007/978-3-030-84060-0_8. URL https://doi.org/10.1007/978-3-030-84060-0_8. (Cited on page 161.)

[277] F. Yates. Contingency tables involving small numbers and the $\chi-2$ test. *Supplement to the Journal of the Royal Statistical Society*, 1(2):217–235, 1934. ISSN 14666162. URL http://www.jstor.org/stable/2983604. (Cited on page 59.)

[278] Daniel M. Yellin and Gail Weiss. Synthesizing context-free grammars from recurrent neural networks. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 351–369. Springer, 2021. doi: 10.1007/978-3-030-72016-2_19. URL https://doi.org/10.1007/978-3-030-72016-2_19. (Cited on page 159.)

[279] Daniel M. Yellin and Gail Weiss. Synthesizing context-free grammars from recurrent neural networks. *Tools and Algorithms for the Construction and Analysis of Systems*, 12651:351 – 369, 2021. URL https://api.semanticscholar.org/CorpusID:231648035. (Cited on pages 94 and 99.)

[280] Inkwon Yeo and Richard A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 12 2000. ISSN 0006-3444. doi: 10.1093/biomet/87.4.954. URL https://doi.org/10.1093/biomet/87.4.954. (Cited on page 144.)

[281] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020. doi: 10.1109/ACCESS.2020.2983149. URL https://doi.org/10.1109/ACCESS.2020.2983149. (Cited on page 1.)

[282] Charles Elkan Zachary Chase Lipton, John Berkowitz. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015. URL http://arxiv.org/abs/1506.00019. (Cited on pages 28 and 29.)

[283] Zheng Zeng, Rodney M. Goodman, and Padhraic Smyth. Learning finite state machines with self-clustering recurrent networks. *Neural Computing*, 5(6):976–990, 1993. doi: 10.1162/neco.1993.5.6.976. URL https://doi.org/10.1162/neco.1993.5.6.976. (Cited on pages 94 and 160.)

[284] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2921–2929. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.319. URL https://doi.org/10.1109/CVPR.2016.319. (Cited on page 1.)

[285] Amirhossein Zolfagharian, Manel Abdellatif, Lionel C. Briand, Mojtaba Bagherzadeh, and Ramesh S. A search-based testing approach for deep reinforcement learning agents. *IEEE Trans. Software Eng.*, 49(7):3715–3735, 2023. doi: 10.1109/TSE.2023.3269804. URL https://doi.org/10.1109/TSE.2023.3269804. (Cited on pages 153 and 163.)