

Benchmarking of Algorithms
on Extended Finite State Automata

Dissertation

zur Erlangung des Grades eines

Doktors der Ingenieurwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

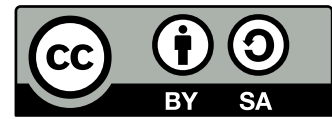
von

Simon Dierl 

Dortmund

2026

This work is licensed under a [Creative Commons “Attribution-ShareAlike 4.0 International”](#) license.



Tag der mündlichen Prüfung: 27.04.2026
Dekan/Dekanin: Prof. Dr. Jens Teubner

Gutachter/Gutachterinnen:
Prof. Dr. Falk Howar
Prof. Dr. Anna-Lena Lamprecht

Abstract

Extended finite state automata are a powerful formalism that can model many complex systems in a finite representation and remain amenable to analysis. A wide variety of algorithms interacts with these automata: for example, automata learning algorithms derive them from black-box systems, while model checking algorithms can verify their properties. When developing such algorithms, it is essential to perform benchmarking, i.e., analyze the algorithm's performance on models of such machines. The choice of benchmark directly impacts the quality of the evaluation, with better benchmarks allowing for well-founded conclusions that are not restricted to, e.g., models of specific types of system.

However, sources for benchmarks for extended finite state automata are limited. Only a single benchmark library and only simple synthesis algorithms are described in the literature. It is not clear what, given these restrictive circumstances, a good selection of benchmarks for a study is and to what extent choice of benchmarks ultimately impacts the strength of the study's results.

This thesis addresses this problem by analyzing limits in the availability and use of benchmarks in both extended finite state automata and finite state automata research and identifying the causes of these limits. We then address these causes and improve upon the current state of the art in benchmarking by defining two synthesis algorithms for extended finite state automata. We also tackle the lack of benchmark libraries by introducing recombinable benchmarks and tools, which simplify the creation and reuse of such libraries.

By analyzing prior works and reflecting on the benchmarking performed in four case studies, we argue that the optimal approach to benchmarking is the combination of synthetic automata with well-curated, constantly-updated benchmark libraries. We successfully apply two novel synthesis algorithms in case studies and show that their benchmarks enable well-founded, strong conclusions. Finally, we retroactively analyze our approach to recombinability on our case studies and show its practicality in terms of effort required.

Keywords Automata Synthesis • Benchmarks • Empirical Content • Evaluations • Extended Finite State Automata • Finite State Automata • Metascience • Register Automata • Timed Automata

List of Papers

- I **Simon Dierl** and Falk Howar. “A Taxonomy and Reductions for Common Register Automata Formalisms.” In: *Model Checking, Synthesis, and Learning*. Ed. by Ernst-Rüdiger Olderog, Bernhard Steffen, and Wang Yi. Lecture Notes in Computer Science 13030. Cham: Springer International Publishing, 2021, pp. 186–218. ISBN: 978-3-030-91384-7. DOI: [10.1007/978-3-030-91384-7_10](https://doi.org/10.1007/978-3-030-91384-7_10)

All proofs and reductions in the paper were developed by me. I am lead author on all sections of the paper.

- II **Simon Dierl** and Falk Howar. “REACH on Register Automata via History Independence.” In: *Tests and Proofs*. Ed. by Laura Kovács and Karl Meinke. Lecture Notes in Computer Science 13361. Cham: Springer International Publishing, 2022, pp. 11–30. ISBN: 978-3-031-09827-7. DOI: [10.1007/978-3-031-09827-7_2](https://doi.org/10.1007/978-3-031-09827-7_2)

The basic algorithmic idea was developed in discussions between both authors. Elaboration of the algorithm, implementation and evaluation were performed by me. I am lead author on all sections of the paper.

- III **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: *NASA Formal Methods*. Ed. by Kristin Yvonne Rozier and Swarat Chaudhuri. Lecture Notes in Computer Science 13903. Cham: Springer Nature Switzerland, 2023, pp. 104–121. ISBN: 978-3-031-33170-1. DOI: [10.1007/978-3-031-33170-1_7](https://doi.org/10.1007/978-3-031-33170-1_7)

Initial analysis of the data was performed and various failed approaches were attempted in full collaboration between all authors over a longer time period. W.r.t. the approach described in the paper, the k -tails minimizer and the TkT baseline implementation were developed by me. I contributed to the Paull-Unger minimizer. I am lead author on Section 5 and co-author of the other sections of the paper.

- IV **Simon Dierl**, Paul Fiterau-Brostean, Falk Howar, Bengt Jonsson, Konstantinos Sagonas, and Fredrik Tåquist. “Scalable Tree-Based Register Automata Learning.” In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Bernd Finkbeiner and

Laura Kovács. Lecture Notes in Computer Science 14571. Cham: Springer Nature Switzerland, 2024, pp. 87–108. ISBN: 978-3-031-57249-4. DOI: [10.1007/978-3-031-57249-4_5](https://doi.org/10.1007/978-3-031-57249-4_5)

Simon Dierl, Paul Fiterău-Broștean, Falk Howar, Bengt Jonsson, Konstantinos Sagonas, and Fredrik Tåquist. “ SL^λ : A Scalable Algorithm for Register Automata Learning.” In: *Journal of Automated Reasoning* 70, 14 (Aug. 2026), p. 14. ISSN: 1573-0670. DOI: [10.1007/s10817-026-09758-9](https://doi.org/10.1007/s10817-026-09758-9)

The conference version was awarded the *Best ETAPS Paper* award by the EATCS.

The work on the SL^λ learning algorithm was done by co-authors. I designed and implemented the generator for all scaling benchmark cases. I am lead author of most of Section 6 of the conference version, Section 8 of the journal version, and co-author of the other sections of the paper.

- V **Simon Dierl** and Falk Howar. *Towards a Specification for Recombinable Benchmarks and Software Tools*. ing.grid Preprint. Universitäts- und Landesbibliothek Darmstadt, July 14, 2025. URL: <https://preprints.inggrid.org/repository/view/55/>. Submitted

The paper is under review at ing.grid.

The underlying motivation stemmed from discussions between both authors. The proposed specification and reference implementation were developed by me. I conducted both case studies. I am the lead author on all sections of the paper.

- VI **Simon Dierl**, Nastaran Kianersi, Falk Howar, and Sean Kauffman. “Learning Probabilistic Automata from Single Continuous-Valued System Logs.” In: *NASA Formal Methods*. Ed. by Jyotirmoy Deshmukh, Klaus Havelund, and Alessandro Pinto. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2026, pp. 170–191. ISBN: 978-3-032-28079-4. DOI: [10.1007/978-3-032-28079-4_8](https://doi.org/10.1007/978-3-032-28079-4_8)

The basic design of the learning pipeline was developed in discussions between all authors. The event selection system, revision score metric, and extension to the IO-ALERGIA algorithm were developed by me. I also performed the evaluation of the metric. I am the lead author on Sections 2, 3.3, 4, 5, and 7.1 and co-author of the other sections of the paper.

Other Peer-Reviewed Contributions

- Ulrich Thomas Gabor, **Simon Dierl**, and Olaf Spinczyk. “Spectrum-Based Fault Localization in Deployed Embedded Systems with Driver Interaction Models.” In: *Computer Safety, Reliability, and Security*. Ed. by Alexander Romanovsky, Elena Troubitsyna, and Friedemann Bitsch. Lecture Notes in Computer Science 11698. Cham: Springer International Publishing, 2019, pp. 97–112. ISBN: 978-3-030-26601-1. DOI: [10.1007/978-3-030-26601-1_7](https://doi.org/10.1007/978-3-030-26601-1_7)
- **Simon Dierl**, Falk Howar, Malte Mues, Stefan Naujokat, and Till Schallau. “Do Away with the Frankensteinian Programs! A Proposal for a Genuine SE Education.” In: *2021 Third International Workshop on Software Engineering Education for the Next Generation (SEENG)*. Institute of Electrical and Electronics Engineers, May 2021, pp. 26–30. ISBN: 978-1-6654-3196-5. DOI: [10.1109/SEENG53126.2021.00012](https://doi.org/10.1109/SEENG53126.2021.00012)
- Malte Mues, Falk Howar, and **Simon Dierl**. “SPouT: Symbolic Path Recording during Testing - a Concolic Executor For the JVM.” in: *Software Engineering and Formal Methods*. Ed. by Bernd-Holger Schlingloff and Ming Chai. Lecture Notes in Computer Science 13550. Cham: Springer International Publishing, 2022, pp. 91–107. ISBN: 978-3-031-17108-6. DOI: [10.1007/978-3-031-17108-6_6](https://doi.org/10.1007/978-3-031-17108-6_6)
- Leon Schwarzer, Matthias Zeller, Daniel Casado Herraiez, **Simon Dierl**, Michael Heidingsfeld, and Cyrill Stachniss. “Self-Supervised Moving Object Segmentation of Sparse and Noisy Radar Point Clouds.” In: *2025 IEEE 28th International Conference on Intelligent Transportation Systems (ITSC)*. Institute of Electrical and Electronics Engineers, 2025, pp. 276–283. ISBN: 979-8-3315-2418-0. DOI: [10.1109/ITSC60802.2025.11423257](https://doi.org/10.1109/ITSC60802.2025.11423257)
- Simon Lutz, Daniil Kaminskyi, Florian Wittbold, **Simon Dierl**, Falk Howar, Barbara König, Emmanuel Müller, and Daniel Neider. “Unsupervised Automata Learning via Discrete Optimization.” In: *Logics in Artificial Intelligence*. Ed. by Giovanni Casini, Besik Dundua, and Temur Kutsia. Cham: Springer Nature Switzerland, 2026, pp. 135–153. ISBN: 978-3-032-04587-4. DOI: [10.1007/978-3-032-04587-4_9](https://doi.org/10.1007/978-3-032-04587-4_9)

Acknowledgments

First and foremost, I want to thank my advisor, Falk Howar, for guiding me through the seven years that lead to this thesis. His advice and experience were as invaluable as his encouragement. I am also deeply grateful for the introductions to a vast circle of collaborators, as well as the wonderful research atmosphere cultivated by him in the AQUA Group. My equal gratitude belongs to the other three members of my evaluation commission, Anna-Lena Lamprecht, Daniel Neider, and Jean Christoph Jung, for both their time and insightful feedback on this thesis. I want to thank Anna-Lena in particular for agreeing to serve as an examiner on *extremely* short notice. My deep thanks also belong to my mentor, Ben Herrmann, for his sage advice throughout the PhD process.

I also want to thank all the many, *many* people with whom I had the pleasure to collaborate on research projects, for stimulating discussions, outstanding ideas, and joint hard work. In addition to the co-authors on the various papers listed above, Bernd Flemisch and the DuMu^x team deserve mention for their help in the ReReSo project. My deep thanks also belong to my student RA, Levi Spielmann, for his outstanding work on the constraints library.

Beyond my circle of co-authors, I am grateful to have worked with all of my colleagues in Dortmund. Without their ideas, assistance, company, and, of course, shared breaks over *many* a cup of coffee, my PhD would have been a lonely endeavor indeed. I also want to thank the students I had the pleasure of advising during their thesis and student projects for their clever ideas and hard work.

My heartfelt thanks belong to my friends and family members, who were always there for me: you encouraged me when I doubted myself, advised me when I was uncertain of my path, and bravely sat through many an anecdote about automata learning and recombining benchmarks. Without all of you, this thesis may never have been written. Last, but *definitely* not least, my final words of gratitude belong to my partner Susanna. Your unconditional support of me, your unquestioning belief in me, and your patience with me carried and motivated me throughout my PhD.

Contents

Abstract	iii
List of Papers	v
Other Peer-Reviewed Contributions	vii
Acknowledgments	ix
Contents	xi
1 Introduction	1
1.1 Benchmarking	2
1.2 Research Problems and Contributions	7
1.3 Structure	10
2 Related Work	11
3 RA Benchmark Transformation	15
3.1 Register Automata	15
3.2 Taxonomy and Reductions	18
3.3 Application to Existing Models	26
3.4 Summary	27
4 Benchmarking in Practice	29
4.1 Existing Works	29
4.2 Case Study I: REACH via Backwards Propagation	49
4.3 Case Study II: SL^λ Learning Algorithm	55
4.4 Case Study III: Intersection Controller Learning	60
4.5 Case Study IV: Heat Pump Learning	68
4.6 Discussion	75

5	Benchmark Synthesis	79
5.1	Structural Synthesis	79
5.2	Semantic Synthesis	82
6	Recombinable Benchmarks and Tools	89
6.1	Recombination	89
6.2	Descriptors and Tool Format	91
6.3	Application	92
6.4	Summary	97
7	Summary	99
7.1	Conclusion	99
7.2	Future Work	101
	Bibliography	103

List of Algorithms

Algorithm 5.1	Structural synthesis algorithm for register automata	81
Algorithm 5.2	Structural synthesis algorithm for register automaton transitions . . .	81

List of Figures

Figure 1.1	A register automaton modeling the single-user account management behavior of XMPP	2
Figure 3.1	XMPP registration in all data universe variants	18
Figure 3.2	Modified XMPP registration and login in all register availability variants .	20
Figure 3.3	Modified XMPP account management in both update granularity variants	22
Figure 3.4	Modified XMPP login in guard-update models admitting duplicate values	23
Figure 3.5	XMPP registration step in guard-update models forbidding duplicate values	24
Figure 3.6	The XMPP register transition in the initialized finite memory formalism, with redundant parts omitted	28
Figure 4.1	Evolution of benchmark types in FSA research	31
Figure 4.2	Evolution of benchmark types in EFSA research	42
Figure 4.3	Modified XMPP operations with and without history independence . . .	50
Figure 4.4	Backwards vs. forwards propagation performance on Automata Wiki data structure automata	52

Figure 4.5	Backwards propagation performance on structurally synthesized automata	54
Figure 4.6	Learning time required by SL^λ , SL^{CT} , and SL^* on the DTLS case study . . .	60
Figure 4.7	Resets generated by SL^λ , SL^{CT} , and SL^* on the DTLS case study	61
Figure 4.8	Resets generated by SL^λ , SL^{CT} , and SL^* on semantically synthesized automata	62
Figure 4.9	A MM1T for the UPPAAL brick sorter example	64
Figure 4.10	Learning setups for reverse-engineering the intersection controller	66
Figure 4.11	A CTMC for a biased coin flip	69
Figure 4.12	Learning pipeline for reverse-engineering CTMCs from arbitrary systems	71
Figure 4.13	subCSL vs. revision scores on synthetic automata	74
Figure 5.1	Graph representation of the guard $(v_1 = v_2) \wedge (v_2 = v_3) \wedge (v_3 \neq v_4) \wedge (v_4 = v_5)$	80
Figure 5.2	Examples of combinator application	85
Figure 6.1	A recombinable learning tool and benchmark are integrated for an evaluation	90

List of Listings

Listing 5.1	The XMPP register automaton defined via the RAGe DSL	84
-------------	--	----

List of Tables

Table 4.1	Purely theoretical works in FSA research by field	32
Table 4.2	Use of case studies in passive FSA learning	33
Table 4.3	Use of case studies in active FSA learning	34
Table 4.4	Use of case studies in other FSA research fields	34
Table 4.5	Use of toy examples in passive FSA learning	35
Table 4.6	Use of toy examples in active FSA learning	36
Table 4.7	Use of benchmark libraries in passive FSA learning	37
Table 4.8	Use of benchmark libraries in active FSA learning	38
Table 4.9	Use of benchmark libraries in other FSA research fields	38
Table 4.10	Use of synthetic benchmarks in passive FSA learning	39
Table 4.11	Use of synthetic benchmarks in active FSA learning	40
Table 4.12	Use of synthetic benchmarks in FSA distinguishing sequence generation .	40
Table 4.13	Purely theoretical works in EFSA research by field	43
Table 4.14	Use of case studies in passive EFSA learning	43
Table 4.15	Use of case studies in active EFSA learning	44
Table 4.16	Use of case studies in other EFSA research fields	45
Table 4.17	Use of toy examples in EFSA research by field	46

Table 4.18	Use of benchmark libraries in EFSA research by field	47
Table 4.19	Use of synthetic benchmarks in EFSA research by field	48
Table 4.20	Backwards vs. forwards propagation performance on non-family Automata Wiki automata	53
Table 4.21	Time required by SL^λ , SL^{CT} , and SL^* on the Automata Wiki	58
Table 4.22	Operations generated by SL^λ , SL^{CT} , and SL^* on the Automata Wiki	59
Table 4.23	Accuracy and result automaton size of all intersection learning pipelines	68
Table 4.24	Best revision scores on non-synthetic models	75
Table 6.1	Trace benchmarks libraries converted into a recombinable format	92
Table 6.2	Artifacts created and code required for recombinability case studies	94

Chapter 1

Introduction

When interacting with a computer system controlled by a computer, we often want to understand its behavior to ensure that it meets our expectations regarding, e.g., correctness. However, to perform the necessary tests, we require both access to the system and a way to analyze it. For example,¹ assume we are setting up an account on an XMPP-based [Sai11] chat server. On a basic level, we want to be sure that after we have created an account, we can then log in using our chosen username-password combination. Since the software is running on a remote server and may only be available as a compiled binary, we lack both access and an easily-analyzable representation.

Automata learning encompasses a set of methods that take a black-box system and derive an automata representation modeling the system’s behavior. This problem was first studied by Gold as “language identification in the limit” [Gol67]. By now, many learning algorithms have been published in the literature, with Angluin’s L^* [Ang87] and Biermann and Feldman’s k -tails [BF72] being seminal works in the field. In our example, the automata learning algorithm should recover a model close to the one shown in Figure 1.1 if the server is implemented correctly.

We usually differentiate *active learning* methods that interrogate the black-box system and *passive learning* methods that infer the system’s behavior by generalizing from a set of *traces* of the system’s inputs and outputs. In our example, an active learning algorithm would actively interact with the server sending and receiving XMPP messages, while a passive algorithm would only require a set of message protocols.

Having learned a model, we can then proceed to analyze it. Depending on the property we are interested in examining, we may select one from a variety of different algorithms. When interested in the correctness of a system, we will usually employ a model checker to verify (or falsify) that a certain property is maintained by the model. In our example, we would verify that the input $\text{register}(u, p)$, $\text{login}(u, p)$ leads to a logged-in location (marked as accepting in the model). Under the assumption that the model is an accurate representation of the

¹This example is based on a running example by Cassel et al. [Cas+15].

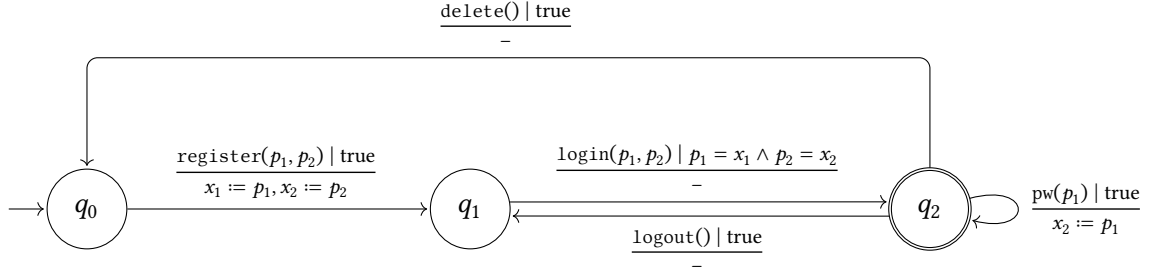


Figure 1.1: A register automaton modeling the single-user account management behavior of XMPP

original black-box system, we can then conclude that the results of the analysis also applies to the original black box. Such approaches to checking properties of black-box systems were named *black-box checking* by Peled et al. [PVY99].

Simple systems can usually be described by automata operating on a finite state space, i.e., a *finite state automaton (FSA)*. On the other hand, systems interacting with humans or cyber-physical systems often operate on an infinite state space to model, e.g., the storage of user inputs (such as username and password in the XMPP example) or the passage of time. Many of these more complex systems can, however, still be described in a finite model as an *extended finite-state automaton (EFSA)* [CK98]. EFSA combine a finite *location* space with an additional data storage. Transitions between locations can depend on, read, and update the data storage. For example, register automata [KF94] form a subset of EFSA that can remember input symbols from an infinite domain (e.g., strings) and check subsequent symbols for (in-)equality. Figure 1.1 shows a register automaton in the succinct formalism defined by Cassel et al. [Cas+15]. In general, most types of EFSA are far more expressive while still remaining algorithmically analyzable, although with higher algorithmic complexity than FSAs.

1.1 Benchmarking

When developing novel algorithms for learning or analyzing automata, evaluation is usually performed empirically by applying the algorithm to a set of *benchmarks*, i.e., automata that are representative of the expected target domain (usually, some class of real-world systems). While some theoretical results w.r.t. to, e.g., complexity bounds are described in the literature, these are usually complemented by an empirical study. For example, an active learning algorithm could be benchmarked by trying to learn the model of a network protocol and counting the number of queries required. However, when evaluating a passive learning algorithm instead, a benchmark would usually include both inputs for the algorithm, i.e., a set of network traces, and a “correct” automaton serving as ground truth to measure the correctness of the learning result. We can formalize this term:

Definition 1 (Benchmark). Assume that we want to evaluate one or more algorithms in a study. We therefore select a set of measurements we want to conduct. A *benchmark* usable for that combination of algorithms and measurements combines

- an input that can be used to execute all algorithms and
- additional data needed to conduct the desired measurements.

Historically, the term described reference programs used to judge system performance [Jos65]. Lewis and Crews give a wider perspective on the history of benchmarking [LC85]. These historic definitions also consider benchmarks as sets of inputs to drive measurements, however, since they only consider timing measurements, they do not consider the need for additional data to perform an evaluation. The term “benchmark” is used in other domains as well, e.g., in business [Cam24]. We solely focus on its meaning w.r.t. computing.

The results of an evaluation using only a single benchmark would be very limited: we could only support claims about the performance of the algorithm in this specific case with confidence. Were we to add more network protocol benchmarks (i.e., a *representative* selection), we can generalize the result to *all* network protocols. We can repeat this step for models of different systems, until we might arrive at an unqualified statement about the algorithm’s performance, i.e., predict its performance in *all* scenarios.

This intuition is formalized by Popper as *empirical content* [Pop02]: the more predictions a theory (here, the claimed results of a study) makes, the better it is. The “amount” of predictions is in turn influenced by the qualifications (i.e., constraints to applicability) attached to the result. In our example, the result’s empirical content would increase not primarily with the number of benchmarks, but their diversity. Applied to the choice of benchmarks, this means that a good benchmark selection is one that results in a high empirical content of the study’s results.

Empirical content is related to the notion of external validity [Win99] (cf. Shadish et al.’s textbook [SCC02]). However, external validity focuses on the correspondence between *claimed* and *actual* empirical content when generalizing results – a result can be externally valid, but can have low empirical content. A lack of the former results in an unsound result, the latter may only be undesirable.

In this thesis, we consider – similar to Stol and Fitzgerald’s *ABC framework for Software Engineering research* [SF18] – five categories of automata benchmark. Each category uses different types of data and can support different types of evaluation with varying threats to validity. The first three categories are benchmarks that are manually defined or curated.

Case Study When research is performed to address a particular use case (e.g., to learn and analyze a model for a specific piece of hardware), benchmarking on the use case is a natural choice. In other works, realistic applications are explicitly sought out. We call such benchmarks *case studies*. While these automata are representative for systems in

the target domain, they also impose inherent limits: first, results from a single case study may not generalize well, second, they may masquerade “overfitting” of an algorithm to the case study, and third, depending on the case study, a ground truth – i.e., a “correct” automaton – may not be available at all, precluding some forms of evaluation altogether.

We define case studies to be benchmarks that either model or are real-world systems that address a realistic issue outside the control of the researcher conducting the evaluation. Crucially, they inherit the complexity characteristic of the real world without any interference by the researcher. Our definition is similar to that of Runeson et al. [Run+12, Section 2.2]. However, Runeson et al. use the term for an *entire* study, while we only consider a single input. As a result, we do not require “multiple source of evidence”, but agree that a case study analyzes “a contemporary [...] phenomenon within its real-life context”. In addition, we focus on the lack of influence over the investigated system, while Runeson et al. do not.

We can divide case studies into several subcategories depending on the exact nature of the case study. In the literature, common types are black-box systems (e.g., a closed-source software or a hardware device), white-box systems (e.g., open-source applications), and models derived from specifications (e.g., network protocols can be formalized into state machines and then used). The amount of information available also influences the degree to which an algorithm can be adapted to a case study.

An example of a case study on a closed-source system is Aarts et al.’s work [ASV10] on a novel learning approach to infer a model of a biometric passport. The algorithm was then benchmarked on said passport. Since the passport is a real-world system outside the author’s influence, we classify this as a case study.

Toy Example Often, an algorithm’s correctness is tested by executing it on small, manually defined examples. While these are frequently inspired by real-world issues, they are usually less complex. Alternatively, such examples may have been developed for didactic purposes or designed to show a particular shortcoming of existing algorithms that is addressed by the authors’ work. We call such benchmarks *toy examples*. They are usually not defined by a domain expert and are not intended to be fully representative of the target domain. In addition, they may be prone to bias if the authors choose models particularly suitable for their work. On the positive side, they can be defined in the target automaton model, enabling analyses not possible with some case studies.

In our taxonomy, toy examples encompass all benchmarks that are either manually defined by the researchers performing the evaluation, even if they are intended to mimic a real-world system, or reuse a selected toy example from another work. While the term is occasionally used in the literature, it is rarely defined. Shaw notes that “toy and textbook examples often fail to provide persuasive validation” [Sha03], while Runeson and Höst mentions “mere toy examples”, setting them apart from case studies [RH09]. Both agree that they are less suitable for evaluation purposes.

Case studies that are simplified by a researcher may occupy a gray area between case studies and toy examples, depending on the nature of the simplification. For example, the technique of removing encryption when learning network protocols will usually not remove any complexity, but this may not be the case for other alterations such as removal of sequence numbers.

Examples include the brick sorter [Ive+00] for UPPAAL [LLN18a; LLN18b], a model of a very simplistic brick sorting robot. While the problem is inspired by a real-world model, the authors have full control over the model and its complexity, making this a toy example instead of a case study.

Benchmark Library In many fields of research, attempts have been made to collect benchmarks into *benchmark libraries*, i.e., collections of benchmarks that were judged to be of interest by human curators. Often, these stem from previous case studies or are adapted from specifications, however, other libraries may be the result of synthesis for e.g., a competition. Use of a well-curated library both ensures some generalization and ensures that the selection of benchmarks is not inadvertently tuned to the study.

Formally, a benchmark library is any selection of benchmarks that was curated and made accessible outside the scope of the study. While they may contain benchmarks of all other categories, the curation process is not in the researcher’s hands. The need for such libraries is well-documented, with Conti’s call for the creation of such a library being an early example from performance benchmarking [Con79].

Internal libraries collected by research groups, such as early versions of the Automata Wiki [Nei+19], may again occupy a gray area, since curators and researchers may easily overlap. Without any outside publication and release process, it becomes difficult to define when a collection of, e.g., case studies becomes a benchmark library.

The Automaton Wiki [Nei+19] is an example of a benchmark library primarily collecting case studies, while the PAUTOMAC benchmark set [VEH14] was synthesized to be used in the eponymous competition.

An alternative approach is the *synthesis* of automata, i.e., the automated generation of a benchmark set matching some predefined criteria. While synthesis includes purely random generation, some synthesis methods add logic beyond mere randomness. In general, two approaches to synthesizing automata-like structures are described in the literature. While in some cases (e.g., passive learning) no automaton is required to run the learner, a meaningful evaluation requires a ground truth (i.e., a canonical automaton corresponding to the inputs). As a result, all synthesis processes observed in our literature survey (cf. Section 4.1) did generate automata

Structural Synthesis Intuitively, automata can be synthesized by creating the automaton’s graph structure, i.e., defining locations and transitions between them according to a rule-based process. Most simply, this can be done by randomly creating transitions over a set

of locations. We call these processes *structural synthesis*. Such techniques are suitable to generate families of automata differing in graph-related properties such as transition density. However, these approaches tend to yield automata defining simple languages, i.e., accept relatively similar inputs without complex semantic features. Additionally, for algorithms that do not directly access the graph structure (e.g., active automata learning algorithms), any complexity solely present in the transition structure can not impact the algorithm at all. This approach is therefore more suitable for algorithms operating on the graph structure.

In our definition, automata generation via structural synthesis includes all algorithms that operate by creating an automaton's graph structure and can be parameterized in terms of graph properties (number of locations, transition density, etc.). A generation campaign using a structural synthesis algorithm could e.g. synthesize k automata with n locations and $2n-4n$ transitions. The language properties of the resulting automata, however, would be emergent; a user can not directly control these during generation.

An example for structural synthesis are techniques based on the random graph generation defined by Erdős and Rényi [ER60]. This algorithm randomly assigns transitions to states, i.e., operates on the graph structure.

Semantic Synthesis Alternatively, we can leverage the fact that automata can be defined by their accepted language. We first synthesize a language that is expressible by our target automaton and then derive a matching automaton. Using this approach, the creation of semantically equivalent automata can be avoided. We therefore categorize these methods as *semantic synthesis* techniques. Here, it becomes possible to vary properties of the automaton's language; usually, the resulting automaton model is a close-to-minimal model for that language. This technique is therefore more suited to algorithms operating on the automaton's behavior, not the graph structure.

For this taxonomy, semantic synthesis algorithms do not operate on an automaton's graph structure. As a result, they can likely be configured using parameters closer to the language's properties. Such properties encompass a wide set of possible parameters, which we do not attempt to restrict for this definition. For example, a synthesis process could be parameterized via the *minimum* number of locations or the nesting level of Kleene stars in an equivalent regular expression. The graph properties of the final automaton are therefore emergent.

An example is the random generation process for DFAs described by Champarnaud and Paranthoën [CP05], which is parameterized number of unique locations, identified by their access strings. While this also allows for some control over the automaton's size, we consider the property to be a semantic one, since it is close to a (subjective) notion of "language complexity".

For various types of finite-state automata, sizable collections of benchmarks exist. Examples include the Automata Wiki [Nei+19] or the RERS competition’s challenges [How+21]². In addition, synthesis of such models has been studied in detail (cf. the survey by Nicaud [Nic14]). As a result, most works developing algorithms on finite-state automata are free to employ all types of benchmark described above.

However, this is not the case for *extended* automata. To the best of our knowledge, only the Automata Wiki’s collection of succinct register automata exists, while for timed models, no collection is available. For synthesis of EFSAs, only random approaches are described in the literature and are used sparingly. As a result, empirical evaluations of EFSA algorithms are frequently performed on a narrow set of benchmarks and benchmark categories. In our informal survey of the literature³, less than 10 % of the works used a benchmark library, ca. 15 % used synthesis, while almost half used only one or more case studies. Of the works using *only* case studies, slightly less than 50 % base their findings on only a single case study (e.g., Aarts et al. [ASV10] consider only a biometric passport). In recent works, the use of multiple benchmarks has become more common (e.g., Vaandrager et al. [VBE21] evaluate on four case studies, two of which are sourced from a previous work.) and increasingly, studies combine multiple categories of benchmark.

1.2 Research Problems and Contributions

We will now describe the research problems addressed by the thesis and summarize the contributions made. Considering the lack of readily-available benchmarking options available for algorithms operating on EFSAs, our motivating question is:

*How can we benchmark algorithms operating on
extended finite state automata to obtain results
with high empirical content?*

This thesis focuses on four research problems, each of which contributes to answering this question in part.

RA Benchmark Transformation As discussed above, the only benchmark library containing EFSAs is the Automata Wiki’s [Nei+19] collection of register automata. The collected automata use the succinct register automaton formalism [Cas+15], a highly expressive variant of register automata. However, more restrictive formalisms (e.g., Kaminski and Francez’s seminal definition [KF94]) are described in the literature. To

²We offer a more comprehensive overview in Section 4.1.

³The full survey is introduced in Section 4.1.

benchmark algorithms operating on restrictive formalisms via Automata Wiki benchmarks, we would need to “downcast” the algorithms into the more restrictive formalism. However, most publications only ensure that their formalism of choice subsumes prior ones, but does not defined an inverse mapping. In addition, such a transformation may alter the automaton’s properties or introduce considerable overhead in terms of automaton size.

In [Chapter 3](#), we will survey register automaton formalisms from the literature and define a comprehensive set of transformations between the formalisms. These not only enable the use of the Automata Wiki benchmarks in contexts requiring other formalisms, but allow for transformations between all models surveyed by us. We will also consider the overhead induces by the transformation and show that most reductions in expressiveness carry an exponential overhead, making the use of “downcast” benchmark libraries in benchmarking impractical. These results are based on [Paper I \[DH21\]](#).

Benchmarking in Practice The empirical content generated by a study’s conclusions is highly dependent on the benchmarks employed to support said conclusions. However, little guidance for benchmark use in EFSA research exists, and the current state of the art may not reflect best practices. Identifying such best practices requires multiple steps. First, we need to estimate the impact of benchmark choice on empirical content by examining applications of benchmarking. We can then make recommendations for benchmarks or benchmark combinations that yield a high empirical content.

Where we find the status quo to be unsatisfactory, we can identify inhibitors to the adoption of best benchmarking practices. In addition, we will consider other undesirable trends in benchmarking and speculate on how to counteract them. The results will inform the following to research questions.

In [Chapter 4](#), we therefore survey both FSA and EFSA research to obtain an overview of the state of the art in benchmarking for both fields. We will show that many desirable practices common in FSA research have not yet been adopted in EFSA research. Most notably, use of benchmark libraries and synthesis methods is prevalent in FSA, but not EFSA research, with deprecation of benchmark libraries also being a common phenomenon.

We will then consider four case studies in which algorithms operating on EFSA were developed and evaluated empirically. For each, we will examine the benchmarks used and the results obtained. We will then consider the impact of benchmark choice for the respective study’s empirical content. This is based on [Paper II \[DH22a\]](#), [Paper III \[Die+23a\]](#), [Paper IV \[Die+24; Die+26a\]](#), and [Paper VI \[Die+26b\]](#).

Then, we will summarize the individual results to conjecture that the combined use of benchmark libraries and synthetic automata is desirable for empirical evaluations, but is not prevalent in EFSA research. We will also support our assumption that inhibiting

factors are the lack of bespoke, nontrivial EFSA synthesis algorithms and the difficulty inherent to using EFSA benchmarks.

Benchmark Synthesis To benchmark algorithms on very large sets of automata (i.e., thousands and more), there exists no feasible alternative to synthesis. Synthesis algorithms can create targeted families of automata (e.g., increasing in size or transition density), enabling the targeted evaluation of the scaling behavior of algorithms. While synthesis of “regular” FSAs has been extensively studied, the synthesis of EFSA beyond random generation remains a mostly open problem. In [Chapter 5](#), we describe a structural and a semantic synthesis algorithm for register automata and discuss how to apply them to synthesize other EFSA types.

The former avoids the creation of dead transitions and can be configured to confirm to various automaton properties, e.g., size or guard complexity. This is based on [Paper II \[DH22a\]](#).

Next, we adapt the semantic DFA synthesis algorithm of Champarnaud and Paranthoën [\[CP05\]](#) to generate deterministic register automata. We implement this algorithm in the RAGe tool that allows for the automated combination of synthesized automata with manually-defined fragments to enable the synthesis of register automata with user-defined substructures. This contribution is based on [Paper IV \[Die+24; Die+26a\]](#).

Recombinable Benchmarks and Tools Both increasing the number of EFSA benchmarks curated in libraries and being able to execute them on competing tools is key to more comprehensive evaluations. However, the collection of benchmarks is often inhibited by the difficulty of using a given benchmark with a novel software tool. Conversely, the use of existing tools with novel benchmarks is also a largely unsolved problem in automata learning and analysis.

To address this issue, a standardized interface between benchmarks and software is required. In [Chapter 6](#), we introduce the concept of *recombinable* tools and benchmarks to establish a path towards the development of large and easily-usable benchmark sets and ease evaluations against competing algorithms. This is based on [Paper V \[DH25\]](#).

In summary, we will both outline the limits of currently available benchmarks and show a path forward. For the former, we show the limits of existing register automata benchmarks w.r.t. reductions in expressiveness and the limits of benchmarking in EFSA algorithm research. For the latter, we define synthesis methods for register automata and sketch an approach to define tools benchmarks that can easily be used in downstream research.

1.3 Structure

In [Chapter 2](#), we will give an overview over related work w.r.t. the individual contributions of this thesis. [Chapter 3](#) tackles the first research question. We define transformations between register automaton formalisms and show that in general, they are not feasible in terms of overhead. We address the second research question in [Chapter 4](#) by performing both a literature review and a retrospective on four case studies. We identify trends and limits in both FSA and EFSA benchmarking, define good benchmarking approaches, and outline the contributions needed to address the current limitations in EFSA benchmarking. [Chapter 5](#) describes the first contribution towards this goal by defining two synthesis algorithms for register automata, one structural and one semantic. The second contribution is presented in [Chapter 6](#), in which we introduce the concept of recombability as an avenue to highly-reusable benchmarks and algorithm implementations. We show that the concept can be easily adopted and may help in the creation of EFSA benchmark libraries. Finally, [Chapter 7](#) will summarize the results and contributions of this thesis and provide an outlook on future work.

Chapter 2

Related Work

In this chapter, we will consider related work. For works related to the algorithmic contributions of [Papers II to IV](#) and [VI](#) [[DH22a](#); [Die+23a](#); [Die+24](#); [Die+26a](#); [Die+26b](#)], we refer to the respective publications. For an overview of FSA and EFSA algorithm research in general, we refer to the surveys in [Section 4.1](#).

Benchmarking Jackson et al. develop a guideline for experimental evaluations [[Jac+90](#)]. They differentiate between different types of test sets (i.e., benchmarks) and note potential deficiencies of both libraries and synthetic data. Also, they consider the relation between a paper’s claims and the necessary experiments. The same notion motivates our introduction of empirical content. Seepersad et al. also consider the problem of generalizing results in Design Science Research [[See+06](#)]. In their Validation Square process, they consider that while support for generalization can be built experimentally, it ultimately has to be a “leap of faith.”

Benchmarking and research design in (Empirical) Software Engineering has been studied with a focus on human experimentation, such as usability studies. Perry et al. discuss the use of empirical studies and disambiguate some types of studies, including studies based on synthetic data [[PPV00](#)]. Sim et al. discuss benchmarking as a technique [[SEH03](#)]. They compare benchmarks to case studies with human subjects, noting they are similar in lack of researcher control over the scenario, but add the ability to replicate results. Empirical methodologies are also discussed in passing in Shaw’s tutorial [[Sha03](#)] and Runeson et al.’s guidelines on case studies [[Run+12](#)]. Stol and Fitzgerald’ *ABC of Software Engineering research* [[SF18](#)] provides both a taxonomy, an overview of approaches, and a survey on related literature. However, this taxonomy cannot be applied to algorithm research as-is. Several of the core factors influencing human participants (e.g., lab setting versus observation studies) are irrelevant to benchmarking of algorithms. Our taxonomy is therefore inspired by the ABC, but can not simply adapt it.

In other fields of Informatics, benchmarks are frequently considered to be a desirable evaluation method. In Machine Learning, benchmarks are both commonly used and the subject of

ongoing research: Thiyaalingam et al. gives an overview over benchmarks [Thi+22], while Hardt seek to develop a “science of machine learning benchmarks” to address shortcomings of current, more ad hoc approaches [Har26]. In Robotics, Amigoni et al. investigate the use of competition data as benchmarks [Ami+15]. More recently, Mendling et al. provide an in-depth overview of Algorithm Engineering methodologies, including empirical work using benchmarks [Men+25]. They stress the importance of selecting appropriate benchmarks, but do not attempt to taxonomize them.

Outside of Informatics, Pingué et al. discuss the need for benchmarks in Engineering Design research [Pin+24]. In Catalysis Science, Bligaard et al. also consider benchmarks a benefit for both computer-based and experimental research [Bli+16]. As a final example, Sharma and Raju survey benchmarks for metaheuristic optimization methods and collect a large selection of well-established benchmark problems [SR24].

Recently, Aichernig et al. performed a large-scale retroactive benchmark of various active FSA learning algorithms [ATW24]. While not explicitly considering the impact of benchmark choice, the study nonetheless improves upon previous evaluation results and shows that the “best” algorithm is (partially) determined by the setting and oracle used, i.e., no clear ranking can be made. This highlights the importance of comprehensive benchmarking. A similar observation for optimization algorithms was made by Fister et al.: even well-established benchmark libraries did not yield consistent results in identifying the best available algorithm [Fis+21]. In the field of security research, Kouwe et al. identify a set of “benchmarking flaws” that may threaten a study’s results’ validity, including selective choice of benchmarks or the use of benchmarks inappropriate for the study [Kou+19]. They note that even high-impact flaws are widespread in the literature, highlighting a need for best practices.

For a discussion of the limits of benchmarking vs. theoretical results in Informatics, we refer to the pair of papers on the subject by Mitzenmacher [Mit15] and Ullman [Ull15]. Kriegel et al. consider possible pitfalls in empirical studies, including limitations imposed the benchmark sets used and unsound comparisons resulting from badly-implemented comparators [KSZ17].

For a general overview of benchmarking, we refer to Lewis and Crews’s survey for an overview over benchmarking history [LC85]. For more recent works, we refer to Zhan’s call for establishing a science of benchmarking [Zha21] and Zhan et al.’s attempt to formalize of “Evaluatology” and “Benchmarkology” [Zha+24]. Both works pursue a holistic approach in describing evaluations and benchmarking, but do not attempt to taxonomize benchmarks in a manner comparable to ours.

Register Automata In the works introducing their respective formalisms, Neven et al. [NSV04], Segoufin [Seg06], Demri and Lazić [DL09] show or assert equivalence of their RA formalism to the seminal finite memory formalism [KF94]. Murawski et al. introduces the notion of *register disciplines* [MRT17], which categorizes formalisms into the same complexity groups as our taxonomy. However, transformations are never specified in detail. Most notably,

none of the works specifies translations into the finite memory formalism. For further reading on the subject, we refer to the general surveys on register automata by Segoufin [Seg06] and Kara [Kar16].

Algorithms on Automata While not the subject of the thesis, we consider a large selection of works introducing or evaluating algorithms that either learn or analyze automata in Section 4.1. In addition to that section, we refer to the vast amount of surveys on the subject, shown here in chronological order. For a survey of learning algorithms defined prior to the seminal L^* algorithm [Ang87], see Angluin and Smith’s survey [AS83]. Next, an overview of theoretical results was given by Pitt [Pit89]. Murphy surveys algorithms that are mostly of historical interest [Mur95].

Later surveys give an overview over current approaches. Balcázar et al. surveys active learning algorithms [Bal+97]. Higuera [Hig05] and Leucker [Leu07] provide more recent overviews, with the former focusing on grammar learning and the latter on learning and verification. Higuera’s textbook gives a broad overview over the field [Hig10]. For an introduction into applying active learning algorithms, we refer to Howar et al.’ paper [How+12b].

Recent overviews and surveys include Vaandrager’s overview article [Vaa17], Bennaceur and Meinke’s more general overview of machine learning [BM18], Howar and Steffen’s survey of works between 2011 and 2016 [HS18], and the recent surveys by Wang et al. [Wan+20] and Ali et al. [ASZ21].

Automata Synthesis Champarnaud et al. compare two structural synthesis approaches to generate random NFAs, one by creating a random transition function and one that begins with a connected structure [Cha+04]. The former approach tends to result in unreachable locations, the second approach is similar to the structural synthesis employed by us. Some other structural synthesis approaches are also essentially random, but create nontrivial automata. Hanneforth et al. define an random algorithm that nonetheless generates nontrivial automata through a well-designed process [HMQ13]. The generation automaton used for the PAUTOMAC competition also operates using mostly-random process [VEH14]. Héam and Joly define a structural synthesis algorithm for pushdown automata that is biased towards creating useful automata through generation of intermediate structures [HJ15].

Alternative approaches to structural synthesis include sampling random transition systems until one matches the desired criteria [CN12a], using a Markov chain to generate the final automaton, resulting in a uniform distribution of created automata [CD12], and randomly creating an intermediate transducer [HNS10].

Other approaches operate by enumerating intermediate structures. A random number can then be transformed into the structure, which, in turn, it transformed into the target automaton. If these transformations are bijective, the automaton is unique. This results in a semantic

synthesis procedure. An example is Champarnaud and Paranthoën’s DFA generator (using generalized tuples and trees as intermediates) [CP05], which forms the basis of our semantic synthesis framework. Other works in this family generate DFAs via boxed diagrams [BN07; BDN08], initially-connected DFAs using a custom string representation [AMR07], unary DFAs via regular expressions [ZR08], and a subtype of NFAs using direct enumeration [FMR18].

For a general overview on automata synthesis, we also refer to Raitt’s thesis [Rai06] and a survey by Nicaud [Nic14].

Recombinable Artifacts Our notion of recombability bears resemblance to *technical interoperability* as defined by the EOSC framework [Dir+21]. The EOSC framework, however, focuses on the exchange of data between systems, while we are interested in the loading of data at rest. Collberg and Proebsting introduce the notion of *repeatability* as the ability to execute research code and find that, for about half of publications presented at a set of 2012 and 2013 conferences, the corresponding research software could not be accessed, compiled, or does not work [CP16].

Examples of attempts at standardizing exchange formats and creating libraries of automata models include the Automata Wiki [Nei+19], AutomatArk [DAn24], as well as the other libraries mentioned in Section 4.1. In other fields, SMT-Lib [BFT25] defines both a standard format for SMT programs and aggregates a large benchmark suite, while the OWASP benchmark [OWA25] is a staple benchmark collection in security research. The SAT Competition [Cod+25] standardizes input and output formats for participating solvers, but does not prescribe a tool format. While LearnLib [FHS25] does not aggregate benchmarks, it collects most algorithms developed for automata learning, i.e., is an example of a *tool* library.

The idea of recombinable tools has roots in the ETI platform [Bra+99], which intended to make research software available over a web interface instead of distributing executable files. A more recent, but field-specific example of recombability is SV-COMP [BS25], which prescribes formats for both benchmarks and tools. The tool format, however, is only designed to work on the competition machines and not any system. In Bioinformatics, the bio.tools project aggregates both data sets and tools, but does not collect tool artifacts, instead linking to artifacts provided by the project developers [Iso+20].

Chapter 3

RA Benchmark Transformation

The Automata Wiki [Nei+19], to our knowledge the only benchmark library containing *extended* finite state automata, expresses its collection of register automata using the succinct RA formalism described used by Cassel et al. [Cas+15]. However, less expressive formalisms, including Kaminski and Francez’s seminal finite-memory formalism [KF94], are also described in the literature. These formalisms may be the subject of algorithms – e.g., Demri and Lazić’s LTL model checking algorithm [DL09] operates on a more restrictive formalism – which cannot be benchmarked with the Automata Wiki benchmarks without converting them to the correct formalism. We will consider if this issue can be addressed by transforming the benchmarks into more restrictive or incompatible formalisms.

In this chapter, we will first introduce the succinct RA formalism in [Section 3.1](#). Then, in [Section 3.2](#), we will define a taxonomy of formalisms by defining four features of register automata and variants of each feature. For each feature, we will propose transformations between all variants and consider their impact on the automata w.r.t. size and semantics. In [Section 3.3](#), we will apply this taxonomy to formalisms from the literature to determine the overhead of transformations between formalisms, i.e., if benchmarks can easily be converted into another formalism. We will show that for almost all formalisms described in the literature, conversion is not a feasible approach. We will summarize the results in [Section 3.4](#).

This chapter is based on [Paper I \[DH21\]](#), but focuses on the reductions’ impact on the automaton, i.e., the benchmarks themselves. For details of the transformations and complexity proofs, we refer to the paper.

3.1 Register Automata

First, we will shortly introduce the syntax and semantics of register automata (RAs). These definitions are based on those used by Cassel et al. [Cas+15]. In [the following section](#), we will introduce more restricted formalisms in terms of these definitions.

Register automata are a type of extended finite state automata that extends FSAs to languages over infinite alphabets (e.g., strings or numbers). However, they are restricted to storing these values in registers and comparing registers and input values for equality. We begin by defining the type of language accepted by register automata.

Definition 2 (Data Universe, Symbol). A *data universe* is a tuple $U = (\Lambda, D, \mathbf{a})$, comprising

- a finite *label alphabet* Λ ,
- a possibly infinite *data alphabet* D , and
- an *arity function* $\mathbf{a} : \Lambda \rightarrow \mathbb{Z}_{\geq 0}$.

A *data symbol* in this universe is of form $\lambda(d_1, \dots, d_{\mathbf{a}})$ for $\lambda \in \Lambda$ and $d_1, \dots, d_{\mathbf{a}} \in D$. In an abuse of notation, we also refer to the set of such symbols as U .

For $\lambda \in \Lambda$, we denote the *parameter set* $\{d_1, \dots, d_{\mathbf{a}(\lambda)}\}$ by P^λ .

For a given $\lambda(d_1, \dots, d_{\mathbf{a}(\lambda)}) \in U$, we define the *parameter valuation* $\mu : P^\lambda \rightarrow \mathcal{D}$ by $\mu(p^i) \mapsto d^i$.

Register automata attach guards to transitions between locations. A guard is an (in-)equality condition over variables. This leads to the following definition.

Definition 3 (Guard). Given a set of *variables* V , the set of *guards over* V is the set of propositional logic formulas over (in-)equalities over V defined by the grammar

$$\mathcal{G}(V) ::= \text{true} \mid \text{false} \mid \mathcal{G}(V) \wedge \mathcal{G}(V) \mid V = V \mid V \neq V.$$

If, for a set D , a guard $g \in \mathcal{G}(V)$ is satisfied by an interpretation $\nu : V \rightarrow D$ under propositional logic rules, we write $\nu \models g$. For example, $\{\nu_1 \mapsto 1, \nu_2 \mapsto 2\} \models \nu_1 \neq \nu_2$.

If, for a guard g' and *any* interpretation ν , the proposition $\nu \models g \implies \nu \models g'$ holds, g' is implied by g and we write $g \vdash g'$. For example, $\nu_1 = \nu_2 \wedge \nu_2 \neq \nu_3 \vdash \nu_1 \neq \nu_3$.

Finally, we can define register automata themselves. An RA is based on an FSA-like structure with a finite number of locations and transitions, but adds registers to store variables from the data alphabet and extends transitions to store and compare values from the data alphabet.

Definition 4 (Register Automaton). Given an *input data language* $U = (\Lambda, D, \mathbf{a})$, a *register automaton* is a tuple (Q, X, S_0, Q^+, Γ) comprising

- a set of *locations* Q ,
- a set of *registers* X ,
- an *initial state* $s_0 = \langle q_0, \chi_0 \rangle : Q \times (X \rightarrow D \cup \{\#\})$ for an empty value $\# \notin D$,
- a set of *accepting locations* $Q^+ \subseteq Q$, and
- a set Γ of *transitions* $\langle q, \lambda, g, u, q' \rangle$, each defining

- a *source location* q ,
- a *label* $\lambda \in \Lambda$,
- a *guard* $g \in \mathcal{G}(P^\lambda \cup X)$,
- a *register update* $u : X \rightarrow P^\lambda \cup X$, and
- a *target location* q' ,

which we conventionally render as

$$\frac{\lambda(p_1, \dots, p_{a(\lambda)}) \mid g}{x_1 := u(x_1), \dots, x_{|X|} := u(x_{|X|})},$$

with self-assignments being implicit, i.e., we omit assignments of form $x_i := x_i$.

The *states* of the automaton are defined by the current location and the *register valuation*, i.e., the mapping from registers to data values. Formally, the set of states is $S = Q \times (X \rightarrow D \cup \{\#\})$.

Semantically, a register automaton operates on values from the data universe U . For each state $\langle q, \chi \rangle \in S$ and input symbol $\lambda(d_1, \dots, d_{a(\lambda)})$, we define the *current valuation* $v : (X \cup P^\lambda)$ as

$$v(z) \mapsto \begin{cases} \chi(z) & z \in X \\ \mu(z) & z \in P^\lambda \end{cases}$$

We can now define the nondeterministic *transition function* $\delta : S \times U \rightarrow 2^S$ as follows:

$$\delta(\langle q, \chi \rangle, \lambda(d_1, \dots, d_{a(\lambda)})) \mapsto \{\langle q', v \circ u \rangle \mid \langle q, \lambda, g, u, q' \rangle \in \Gamma, v \models g\},$$

i.e., the successors of a state are defined by all outgoing transitions with matching labels where the current valuation satisfies the guard. The new register valuation is computed by applying the update function to “look up” the new value sources and then using the current valuation to resolve them.

We can define a *transitive transition function* δ^* as the repeated application of δ .

Figure 1.1 shows an example for an RA, first used by Cassel et al. [Cas+15, Figure 1], that models the user management subset of XMPP [Sai11] for a single user and accepts if a user is currently created and logged in.

There are two common definitions of determinism for register automata. All demand that each input word induces at most one path through the automaton (this can be strengthened to be *exactly* one by introducing a sink state), i.e., for all $w \in U^*$, $|\delta^*(s_0, w)| \leq 1$. However, they differ w.r.t. their restrictions on guards. Assume two transitions with identical source locations and labels $\langle q, \lambda, g_1, u_1, q'_1 \rangle$ and $\langle q, \lambda, g_2, u_2, q'_2 \rangle$. The stronger definition is mutual exclusion of guards with the same label, i.e., $g_1 \wedge g_2 \vdash \perp$. The weaker definition demands that for all *reachable* states and subsequent inputs, the current valuation $v \models \neg g_1 \vee \neg g_2$, i.e., there is never a choice between transitions. An even weaker constraint is *determinacy*

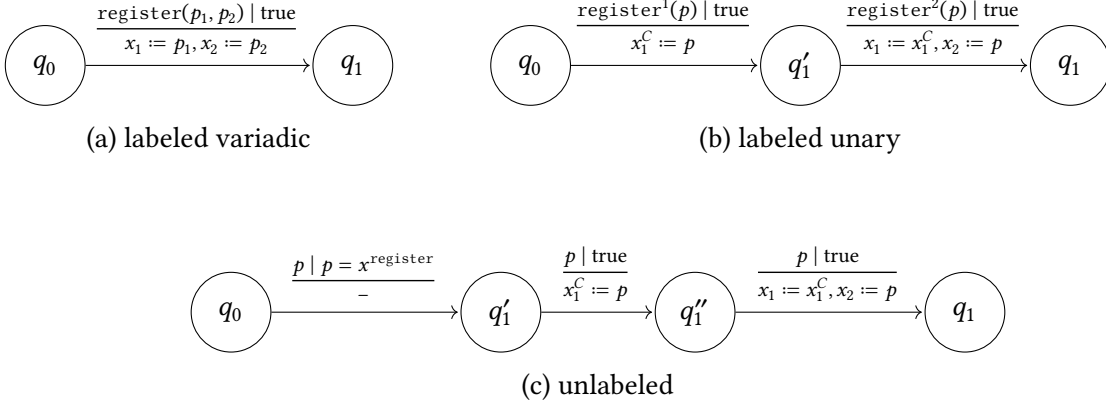


Figure 3.1: XMPP registration in all data universe variants

[Cas+15]. In a determinate register automaton, transitions may be ambiguous, but acceptance behavior must be consistent between all choices. Formally, for every input word $w \in U$, let $Q_w = \{q \mid \langle q, \chi \rangle \in \delta^*(s_0, w)\}$. Then, either $Q_w \subseteq Q^+$ or $Q_w \cap Q^+ = \emptyset$.

All three definitions are semantically equivalent, i.e., strongly deterministic, weakly deterministic and determinate automata can be converted into each other [Cas+15]. However, the set of data languages that can be recognized by deterministic register automata is a strict subset of those recognized by nondeterministic ones [KF94].

3.2 Taxonomy and Reductions

In this section, we will describe the four features we use to taxonomize register automata formalisms. For each feature, we will introduce its variants, sketch transformations between them and outline the impact on the resulting automata. However, in Paper I [DH21], we focused on reductions of the MEMBERSHIP and NONEMPTYNESS problems, i.e., “live” transformations that can operate on both the automaton and inputs.

Here, we focus on conversions “at rest” that keep the automaton “mostly intact”, i.e., the system modeled by the automaton should ideally bear some resemblance to the original and the inputs should be similar to the ones processed by the original automaton. This is motivated by the idea of transforming a benchmark – if the new system is different, we may lose a relevant property or a model’s realism. All transformation we introduce will preserve an automaton’s NONEMPTYNESS property, however, they may require some alterations to the input language. We will discuss these when introducing each transformation.

3.2.1 Data Universe

As the first feature, we will consider restrictions on the automaton's input language. We call this feature *data universe*. When surveying the literature, we identified three variants, examples of which are shown in Figure 3.1 as applied to the XMPP example RA's register transition.

Labeled Variadic The most permissive variant is the one introduced in Definition 2 without any further restrictions. Here, an arbitrary number of labels can be used and their arity can be arbitrary. We therefore call this variant *labeled variadic*. An example is shown in Figure 3.1a.

Labeled Unary Some formalisms define an RA's input language as tuples of a label and exactly one data value, i.e., for all labels λ , the arity $\alpha(\lambda) = 1$. We therefore call this variant *labeled unary*. It defines a strict subset of labeled variadic. Figure 3.1b shows a possible conversion of the register transition into two transitions, each of which reads one of the required data values. The parameters are cached and all guard and assignments are deferred until the last transition.

Unlabeled Unary Some formalisms are defined on the most restrictive variant, in which no label alphabet is used. To match our definition, we assume a single “dummy” label which we omit from the transition notation, and call the variant *unlabeled unary*. This is a strict subset of labeled unary. Since labels must now be read as data values, Figure 3.1c shows the transition decomposed into three steps that read label and the two data parameters.

Transformation from variadic into unary changes the input language of the automaton by splitting symbols with arity > 1 into multiple steps and caching all intermediate values [DH21, Theorem 1]. To preserve determinism, redundant intermediate states are merged. For symbols with arity zero, an unused parameter can simply be added. This conversion results in the addition of at most $\sum_{\langle q, \lambda, g, u, q' \rangle \in \Gamma} \alpha(\lambda) - 1$ new transitions and locations and $\max_{\lambda \in \Lambda} \alpha(\lambda) - 1$ new registers. The input language is changed in a relatively natural fashion: symbols with arity ≥ 2 are split into multiple symbols with arity one.

Removal of the label from a labeled unary RA is more complex: the automaton must define constants for each label [DH21, Theorem 2]. These constants can be set as initial register values. Again, transitions are split, but redundant intermediate states can be merged. The transformation adds at most $|\Gamma|$ new intermediate states. Additionally, $|\Lambda|$ new registers are added. This also makes relatively superficial changes to the input language: instead of labels with data values, the input is now a sequence of interleaved label constants and values.

Transformation into more permissive variants is automatic, since the variants are included in each other.

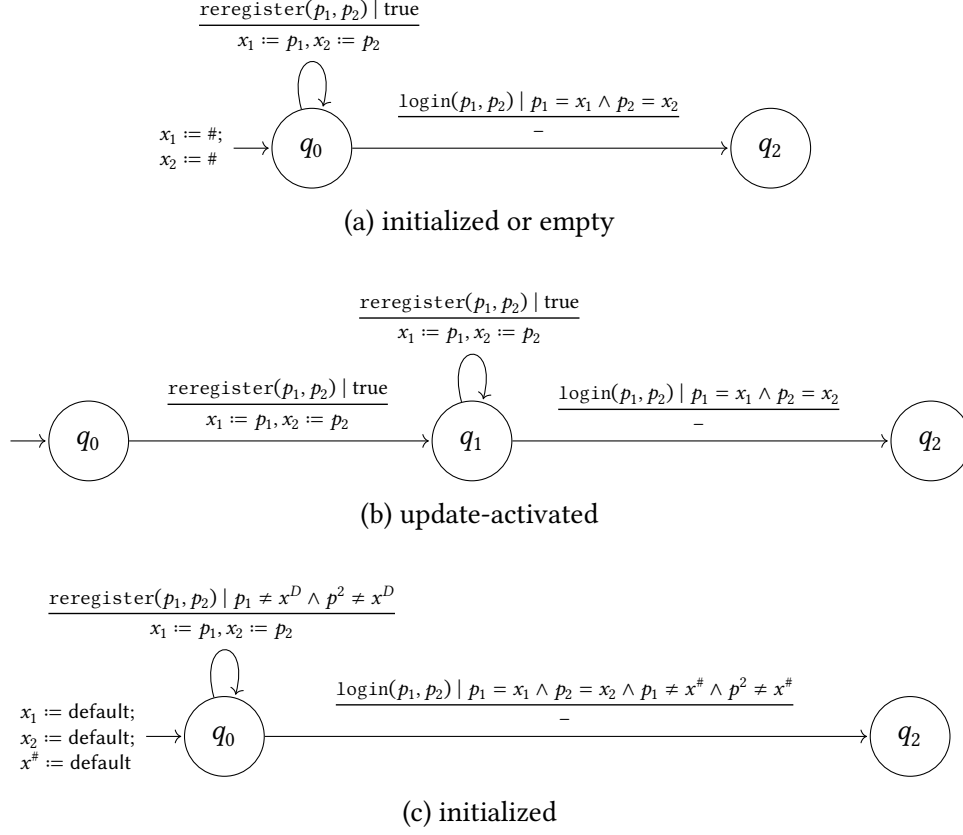


Figure 3.2: Modified XMPP registration and login in all register availability variants

3.2.2 Register Availability

As the second feature, we will consider if and when registers can be used in a guard or update and how the empty value $\#$ may be used. We call this feature *register availability*. When surveying the literature, we identified three variants. As an example, consider a variant of the XMPP example in which the register operation has been replaced with a reregister operation that unconditionally overwrites the previous registration. Logins are only possible if at least one user was registered and the last provided username and password are used for the login. Figure 3.2 shows the resulting reregister and login transitions in each variant.

Initialized or Empty In this variant, registers can be initialized as empty and the empty value can be “seen” by transitions. Note that since $\# \notin D$, it will compare negatively to any possible input value. We call this variant *initialized or empty*. Figure 3.2a shows an example. Note that the login transition can only be taken if the username and password registers are set to a non- $\#$ value.

Update-Activated Here, all registers are initialized to the value # and may only be used in a transition if they have previously been overwritten. More formally, we can define active registers X_q for each location q that may be used in outgoing transitions, with $X_{q_0} = \emptyset$. We therefore usually omit the initialization and call the variant *update-activated*. This is a strict subset of initialized or empty. In Figure 3.2b, we can see that this requires the introduction of an additional location: in q_0 , no registers are active and the login transition may not originate there.

Initialized Under this variant, all registers must be initialized to a data value. We call this *initialized*; it also is a strict subset of initialized or empty. Figure 3.2c shows an example in which we have selected a value default from the alphabet to represent #, store it in $x^\#$, and, during each transition, check for its presence to avoid logins using the default value.

A transformation from initialized to update-activated RAs can be done by adding an initialization to the RA that occurs before the initial location [DH21, Theorem 6]. If registers x_1 and x_2 are assigned to non-equal values, this would result in a transition

$$\frac{\text{init}(p_1, p_2) \mid p_1 \neq p_2}{x_1 := p_1; x_2 := p_2}.$$

All registers can now be initialized to empty; one transition and location are added. Additionally, the input language is changed substantially: each input must begin with the constant values. Additionally, these values now can be changed at “runtime”: each input that retains the original’s relation between values and constants is now accepted by the RA. The reverse operation is simple: the initial valuation can be set to arbitrary values, since all registers are guaranteed to be overwritten before use [DH21, Theorem 3].

Converting initialized or empty to initialized automata employs the technique seen in Figure 3.2c: a dummy value is selected to stand in for # [DH21, Theorem 4]. This adds a single register and increases the complexity of guards. Additionally, it again changes the input language, since the selected dummy value can no longer be used, i.e., inputs including it are no longer accepted. This does not influence the reachability of locations (and therefore NONEMPTYNESS), since the data alphabet is infinite and for each input containing #, an infinite number of alternative inputs with identical behavior can be used.

A conversion from initialized or empty to update-activated automata is possible by chaining both conversion methods. However, this conversion chain is not possible in some restrictive variants introduced in the next section; in Paper I [DH21], we show that in general, an efficient conversion with runtime in NL is impossible unless $\text{NL} = \text{NP}$ [DH21, Theorem 5]. Conversions into initialized or empty are automatic by virtue of inclusion.

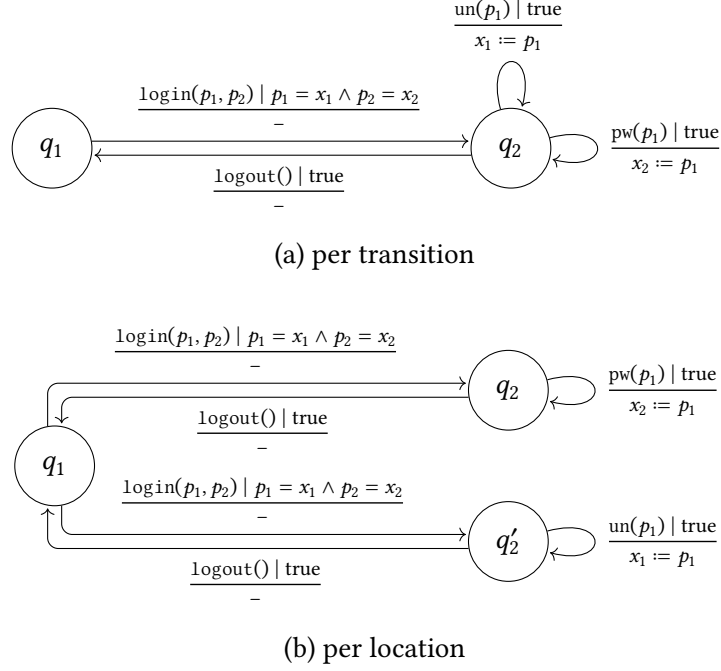


Figure 3.3: Modified XMPP account management in both update granularity variants

3.2.3 Update Granularity

The third feature we consider is the definition of updates. In some formalisms, updates are mandated by the location, not the transition, i.e., each transition must either use the location's update or not change the register valuation. We call this *update granularity*. When surveying the literature, we identified two variants. As an example, assume that in the XMPP example, we can also change our username using the `un` operation when logged in. Figure 3.3 shows the resulting automaton fragment in each variant.

Per Transition In this variant, transitions are free to define their own updates, as described in Definition 4. We call this *per transition*; Figure 3.3a shows an example.

Per Location Under this variant, for each location, there exists a canonical update function u such that each outgoing transition's update must either be the identity function or u . This is a strict subset of per transition, which we call *per update*. Figure 3.3b shows an example: here, we have two logged-in locations, depending on the subsequent modification operation. Logout transitions are available on both, since they do not alter the registers.

Transforming from per transition to per state can be done using two approaches. First, the idea shown in Figure 3.3b, i.e., splitting each location by the outgoing transition's registers, can be applied. While this keeps the input language unchanged, this results not only in a

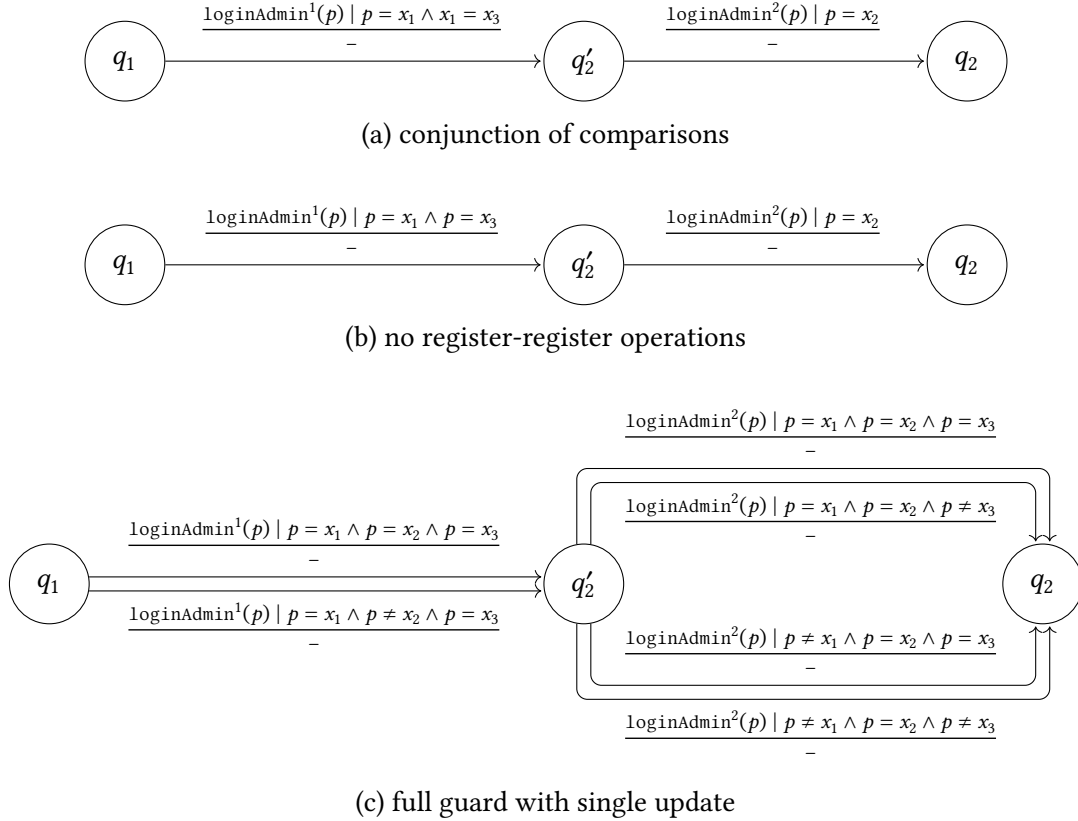


Figure 3.4: Modified XMPP login in guard-update models admitting duplicate values

polynomial increase in size, but also removes an automaton's determinism: in our example, the next input must be guessed when reading `login`.

The second option makes a notable change to the input language by duplicating all inputs and inserting an intermediate location into all transitions [DH21, Theorems 7 and 8]. The first transition will only check label and guard, while the second transition will be a copy of the original one. This requires only up to $|\Gamma|$ new locations and transitions, but substantially changes the input language. However, `NONEMPTYNESS` remains unchanged.

In Paper I [DH21], we show that all transformations that preserve data language and determinism must cause an exponential increase in automaton size [DH21, Theorem 9]. Backwards transformations are again automatic.

3.2.4 Guard-Update Model

Finally, we consider restrictions on the expressiveness of guards and updates. We call this feature *guard-update model*. Here, we identified five models in the literature, with almost

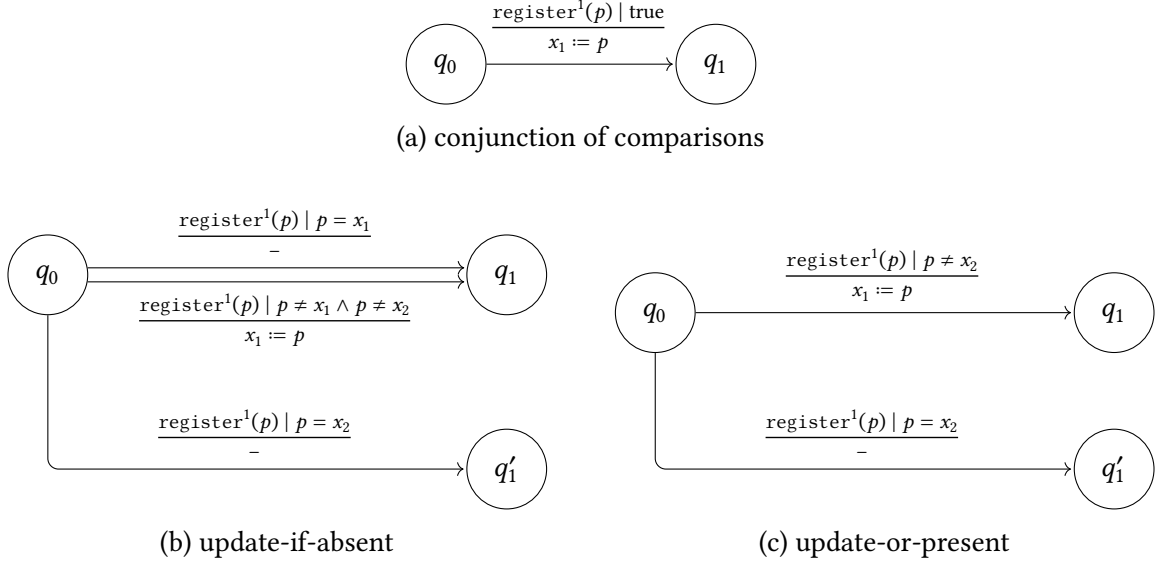


Figure 3.5: XMPP registration step in guard-update models forbidding duplicate values

all formalisms using a custom model. Here, we consider two examples based on the XMPP example: first, a split `loginAdmin` transition in Figure 3.4 that requires the username to match a predefined admin username in x_3 and second, the first half of a split `register` transition in Figure 3.5.

Conjunction of Comparisons In this variant, we place no restrictions on the definition of guards and updates from Definitions 3 and 4. Since the guard may combine comparisons of arbitrary values, we call this variant *conjunction of comparisons*. Examples are shown in Figure 3.4a.

No Register-Register Comparisons Here, we forbid comparisons and assignments between registers, i.e., clauses such as $(x_i = x_j)$ may not be used in a guard and $x_i := x_j$ may not be used in an update. We call this variant *no register-register comparisons*; it is a strict subset of above variant. Note that in the presence of variadic data symbols, the restriction can be circumvented by introducing additional witness parameters to a symbol and checking, e.g., $(x_i = p) \wedge (x_j = p)$ instead. Figure 3.4b shows an example: here, we can not check that the stored username is correct, but can compare the parameter instead (i.e., we use $p = x_3$ instead of $x_1 = x_3$).

Full Guard with Single Update This variant is only defined for unary data universes and demands that during each transition, the parameter must be compared to all registers and may subsequently be stored in a single register. No other updates can be made. We call this *full guard with single update*; it is a strict subset of the second variant. Figure 3.4c shows an example of the downsides: the first transition must be doubled and the second quadrupled to account for all necessary comparisons.

Update-if-Absent This variant ensures that no duplicate values can be stored in registers and is only defined for unary data universes. Here, we can only employ two forms of transition:

$$\frac{\lambda(p) \mid p = x_i}{-} \quad \text{and} \quad \frac{\lambda(p) \mid \bigwedge_{x \in X} p \neq x}{x_i = p}.$$

We call this *update-if-absent*; it is a strict subset of no register-register comparisons. Figure 3.5b shows an example of a conversion from Figure 3.5a: here, we need to account for registrations in which the selected username is already stored in the password register and transition to a different state to remember this fact. Handling an already-stored username is comparatively simple, but requires an additional transition.

Update-or-Present This variant, too, ensures that no duplicate values can be stored in registers and is only defined for unary data universes. Here, we can only employ two slightly different forms of transition:

$$\frac{\lambda(p) \mid p = x_i}{-} \quad \text{and} \quad \frac{\lambda(p) \mid \bigwedge_{x \in X \setminus \{x_i\}} p \neq x}{x_i = p}.$$

We call this *update-or-present*; it too is a strict subset of no register-register comparisons. The example in Figure 3.5c is similar to the previous variant, but we can avoid the duplicate transition for pre-stored usernames.

Transformations in this feature are only meaningful without variadic data symbols. To transform conjunction of comparisons into no register-register comparisons, we can replace each guard of form $x_i = x_j$ with an additional transition with guard $(x_i = p) \wedge (x_j = p)$ and each assignment of form $x_i := x_j$ with a transition of form

$$\frac{\lambda(p) \mid p = x_i}{x_j := p} \text{ [DH21, Theorem 13].}$$

This converts transitions into location-transition chains of length polynomial in $|X|$ and substantially alters the input language by introducing symbol repetitions dependent on the structure of the guards. Still, this method preserves NONEMPTYNESS.

To convert no register-register comparisons into full guard with single update, we can apply the transition duplication technique shown in Figure 3.4c and “fill in” all missing comparisons [DH21, Theorem 14]. While this keeps the input language intact, this results in an exponential increase in the number of transitions.

In Paper I [DH21], we show that no computationally efficient (i.e., polynomial-time) conversion from full guard with single update to update-if-absent can exist unless $\text{NP} = \text{PSPACE}$ [DH21, Theorem 15].

To convert conjunction of comparison to update-or-present, we have to effectively “virtualize” the automaton’s registers [DH21, Theorem 12]. Virtualization information is a mapping from

X to X and describes, e.g., “register x_3 contains the value of x_1 and x_2 , register x_1 that of x_3 .” Using this, we can simulate the effect of register-register assignments, duplicate values, etc. and evaluate register-register comparisons by considering the virtualization information. However, we can only store this information in locations, i.e., we need to duplicate each location for each possible mapping, causing an exponential blowup in locations and transitions. This method does not change the input language.

Converting update-or-present into update-if-absent can easily be done by splitting assigning transitions into a transition for new values and one for already-stored ones, as shown in Figure 3.5b [DH21, Theorem 10]. This can create $|\Gamma|$ transitions at most. For the inverse operation, we can relax guards to not check the target register (since the effect of overwriting a value with itself is non-observable), however, this may make an automaton nondeterministic [DH21, Theorem 16]. This approach, too, preserves the automaton’s input language.

Finally, update-if-absent can efficiently be converted into full guard with single update without altering the language: since no duplicate values can exist, we can extend each guard with the missing inequalities [DH21, Theorem 11]. All remaining transformations now follow from inclusions or can be expressed as conversion chains.

3.3 Application to Existing Models

In this section, we will apply the taxonomy to six formalisms described in the literature and discuss the overhead of conversions between them. The formalisms are:

Succinct RAs are the most permissive formalism [Cas+15], combining labeled variadic data, update-activated registers, per-transition updates, and conjunction of comparisons.

Demri-Lazić automata are less permissive [DL09], combining labeled unary data, initialized registers, per-transition updates, and no register-register comparisons. Note that the automata described by Demri and Lazić must be converted into our formalism.

Segoufin automata are again less permissive [Seg06], combining labeled unary data, initialized registers, per-transition updates, and full guards with single update.

Neven-Schwentick-Vianu automata are very restrictive [NSV04], combining unlabeled data, initialized or empty registers, per-transition updates, and update-if-absent transitions. These, too, require conversion.

Finite-memory automata are the restrictive, seminal formalism by Kaminski and Francez [KF94], combining unlabeled data, initialized or empty registers, per-state updates, and update-or-present transitions.

Initialized finite memory automata are the least permissive [MRT17], combining unlabeled data, initialized registers, per-state updates, and update-or-present transitions.

Using the conversion steps described above, we can “downcast” succinct RAs to Demri-Lazić automata by transforming variadic into unary symbols, selecting random initial values, and

using proxy values to replace register-register actions. This results in polynomial blowup, changes to the input language, and the creation of long transition chains in the automaton structure.

To convert Demri-Lazić to Segoufin automata, we need to create full guards, necessitating the creation of an exponential number of transitions. In [Paper I \[DH21\]](#), we describe an alternative, efficient reduction that preserves some automaton properties, but completely changes the input language.

To transform succinct RAs to finite-memory automata, we need to convert per-transition into per-state updates, sacrificing either determinism, language, or polynomial size, convert variadic to unlabeled symbols, then perform register virtualization to simplify the guards. This inevitably causes exponential blowup and changes the input language.

The conversion of Neven-Schwentick-Vianu automata into finite-memory automata requires converting per-transition into per-state updates and transformation from update-if-absent to update-or-present. This transformation removes determinism.

Finally, transforming finite-memory into initialized finite-memory automata requires the replacement of # with a proxy value. Since the proxy value technique for this requires a rich guard-update model, we have to “upcast” the automaton to conjunction of comparisons and then virtualize registers, causing exponential blowup.

“Upcasting” can usually be performed without drastic changes to the automaton. Initialized finite-memory automata satisfy the constraints for finite-memory automata. These can be converted into Neven-Schwentick-Vianu automata by converting update-or-present to update-if-absent (i.e., splitting transitions). These, in turn, can be converted to Segoufin automata by adding inequalities to all guards and initializing registers to random values. Segoufin automata satisfy the constraints for Demri-Lazić automata, which can be converted into succinct RAs by converting constants into an initialization operation.

3.4 Summary

In summary, the conversion of the succinct models found in the Automaton Wiki [\[Nei+19\]](#) into other formalisms is not feasible in almost all cases. The only conversion that does not cause exponential blowups is that to Demri-Lazić automata. Additionally, even the seemingly simple conversion from Neven-Schwentick-Vianu to finite-memory automata removes determinism, which is a substantial change in model.

While hypothetical benchmarks in a restrictive formalism may be usable in more benchmarking scenarios, this comes at the cost of containing either extremely simple or very large models. In the XMPP example, modeling only registration in the initialized finite-memory formalism would require the automaton partially shown in [Figure 3.6](#). Here, we construct

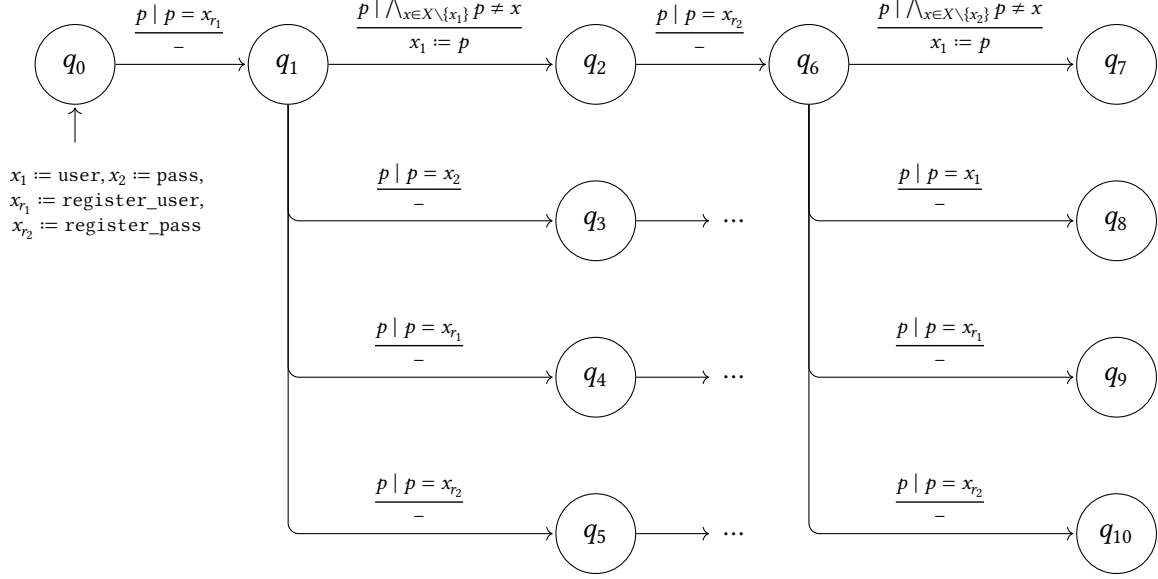


Figure 3.6: The XMPP register transition in the initialized finite memory formalism, with redundant parts omitted

additional locations for each possible value duplication that can occur. It is apparent that in this case, the exponential blowup caused by storing equality information is unavoidable. Such a blowup would, of course, persist after transforming back to a richer formalism.

In conclusion, RA benchmark libraries need to be defined *per formalism* at differing levels of semantic complexity, further restricting the already limited availability of such libraries. While we have not repeated this analysis for other types of EFSAs, this may well be an issue for libraries of those, too. E.g., in general, timed automata [AD94] can not be converted into Mealy machines with one timer [VBE21] at all, since their semantics are not fully expressible in that restricted automaton type.

Chapter 4

Benchmarking in Practice

Benchmarking is an essential step in most research on algorithms operating on (extended) finite state automata, with the choice of benchmarking method influencing the empirical content of the study’s conclusions as described in [Section 1.1](#). In practice, a wide variety of benchmarks is used in the literature, with no clear overview of their impact w.r.t. a study’s empirical content and little guidance to suitable choices for benchmarking. We will therefore first survey the state of the art in benchmarking and then retroactively examine benchmarking in four research projects in detail. This will allow us to identify current shortcomings and make suggestions for improvement.

In the chapter, we will first survey existing publications on both FSA and EFSA algorithms in [Section 4.1](#) to obtain an overview of the adoption of benchmarking methods, the availability and use of benchmark libraries, and the results obtained using the individual methods and their combinations. Next, in [Sections 4.2 to 4.4](#), we will recapitulate the benchmarking performed in support of four research projects developing algorithms operating on EFSA. For each project, we will shortly summarize the background and algorithmic contributions. Then, we will describe the choice of benchmark in detail, discuss the evaluation and its results, and consider the impact of the benchmark method on the results’ empirical strength. We will combine the survey and the case studies’ results in [Section 4.6](#) to obtain a general view of the impact of benchmark selection, current limits of benchmarking in EFSA research, recommendations for method choice, and an outlook.

4.1 Existing Works

To establish an overview on the use of benchmarking techniques outside of the case studies described next, we conducted an informal survey of existing works. As starting points, we used our previous works in automata learning, including those used as case studies in [Sections 4.2 to 4.5](#), the survey papers mentioned in [Chapter 2](#), bibliographies of well-known researchers in the field, and Google Scholar searches for related keywords (e.g., “automata

learning”). From these works, we performed forwards and backwards searches. We ignored works that were not accessible digitally (pre-1990 works, mostly conference papers), as well as publications that did not describe algorithms on automata. In general, our search focused on automata learning, with some consideration for model checking.

For each discovered work, we assigned it to FSA or EFSA research. As a further subdivision, we separate the publications into works on passive learning algorithms, active learning algorithms, and other algorithms (including hybrid passive-active algorithms). Next, we categorized the benchmarks used therein according to our taxonomy. Note that in the following tables, works may occur multiple times, once for each category. We describe works that do not contain any empirical evaluation as (*purely*) *theoretical*. We will first summarize our observations on finite state automaton research to establish a baseline in [Section 4.1.1](#) and then survey works on extended finite state automata in [Section 4.1.2](#).

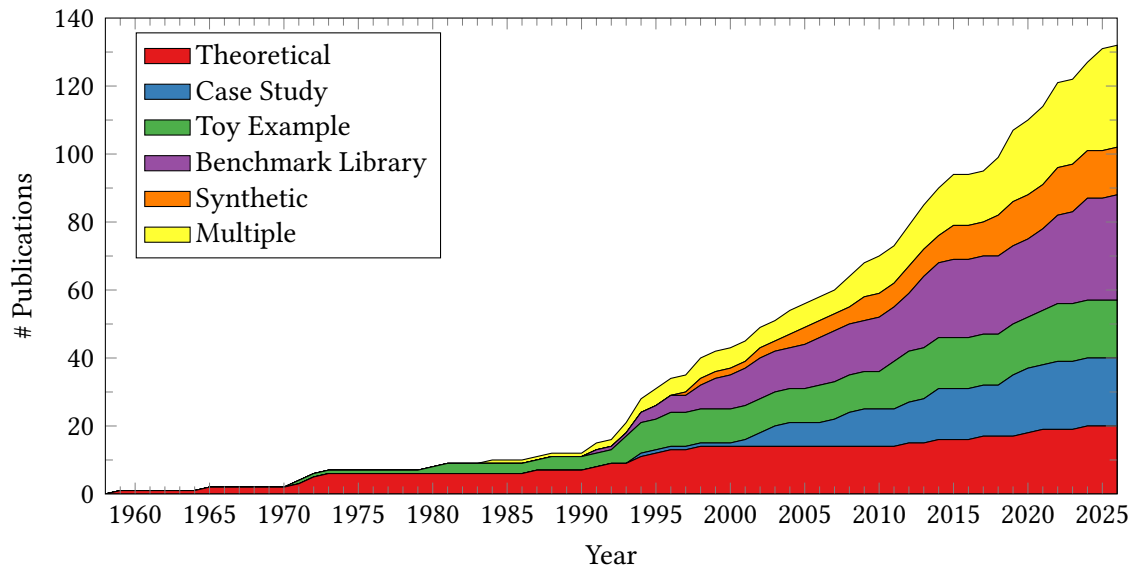
This section contains novel work.

4.1.1 Finite State Automaton Research

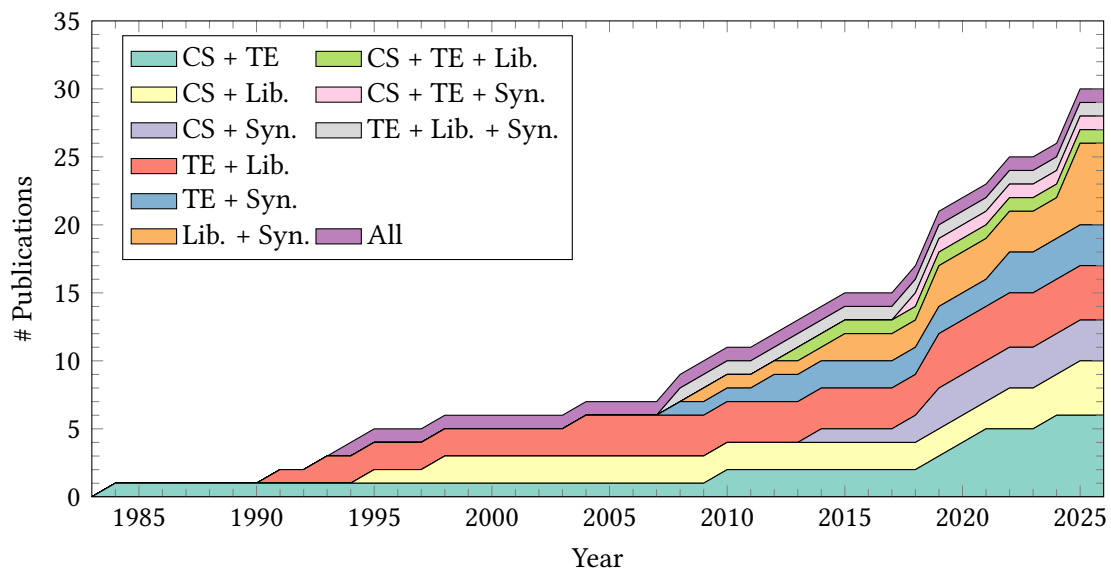
In FSA algorithm research, purely theoretical works usually describe theoretical analyses of algorithms, usually containing correctness proofs or asymptotic runtime bounds. They are common in the seminal works of each sub-field, such as Angluin’s work on L^* for the field of active automata learning. Most recent such publications are focus on a theoretical contribution. E.g., while CALF [HSS17] contains algorithmic implementations, the central contribution is the category-theoretic view on learning. [Table 4.1](#) shows an overview of all works in this group. The work on OSTIA-D [OV96] is an outlier: its predecessor OSTIA [OGV93] was evaluated empirically, while it is only analyzed from a theoretical perspective.

Works using case studies become common around 2000, with passive learning algorithms using this benchmark category slightly earlier. This may be due to the increase in available compute power to conduct such studies, as well as the availability of optimized algorithms that can address complex use cases. In most passive learning publications and later active learning works (ca. after 2010), case studies are usually complemented with other benchmark categories. [Tables 4.2](#) to [4.4](#) show all works classified in this category. The passive learning algorithm ISM [KF94] is an exceptional case: all categories of benchmark were used in its empirical evaluation.

Toy examples are a common feature of early works in passive learning and less common for active learning algorithms. While early toy examples often are bespoke constructions that do not resemble any real-world system, more recent examples usually attempt to model simplified “real” systems. Frequently, toy examples are used in conjunction with other benchmarks. In these studies, the toy examples are often designed to show a novel algorithm’s improvements (e.g., a system that can not be learned with previous approaches), while other benchmarks are employ to add context to this result. For example, Product- L^* [Moe19] outperformed TTT



(a) all types



(b) multiple types (detail view of a)

Figure 4.1: Evolution of benchmark types in FSA research

Year	Algorithm / Authors
1959	Paull et al. [PU59]
1965	Grasselli et al. [GL65]
1971	Kella [Kel71]
1972	k -Tails [BF72]
1973	Pfleeger [Pfl73]
1991	Porat et al. [PF91]
1996	OSTIA-D [OV96]
2024	CALF for EDSM [Vil24]

(a) passive automata learning

Year	Algorithm / Authors
1972	Gold [Gol72]
1987	L^* [Ang87]
1992	Tzeng [Tze92]
1994	Kearns-Vazirani [KV94]
1995	Maler-Pnueli [MP95]
2012	Leucker et al. [LN12]
2014	Jacobs et al. [JS14]
2017	CALF [HSS17]
2020	Generalized L^* [US20]
2021	FunL* [CPS21]

(b) active automata learning

Year	Algorithm / Authors	Field
1994	Lee et al. [LY94]	state identification
1998	IID [PNH98]	hybrid learning

(c) other fields

Table 4.1: Purely theoretical works in FSA research by field

Year	OBT	Algorithm / Authors	Case Studies
1984	T	successor method [RV84]	handwriting recognition
1994		Ahonen et al. [AMN94]	part of a Finnish dictionary
1994	T B S	ISM [Kam+94]	intermediate synthesis steps and outputs of other algorithms
1995	B	Ron* [Ron95]	text correction, DNA, and handwriting recognition
1998	B	Markov [CW98]	industrial change request system
2001		Reiss et al. [RR01]	3 Java and 3 C++ applications of varying complexity
2002		Ammons et al. [ABL02]	17 X11 client programs
2009		Prospex [Com+09]	SMTP [Kle08], SMB [Mic25], and SIP [Sch+02] implementations
2014		Buzhinsky et al. [Buz+14]	maneuvers in a flight simulator
2018	S	Avellaneda et al. [AP18]	flight simulator components
2018	T S	Ulyantsev et al. [UBS18]	2 Java applications
2019	T	App. IOALERGIA [AT19]	MQTT [OAS19], TCP [Edd22], and the shared coin consensus protocol [AH90]
2019		App. k -Tails [GDS19]	hardware driver
2020	T	Jeppu et al. [Jep+20]	3 emulator components and the Linux kernel

OBT: other benchmark types used **T:** toy examples **B:** benchmark libraries **S:** synthesis

Table 4.2: Use of case studies in passive FSA learning

Year	OBT	Algorithm / Authors	Case Studies
2003		L_{opt}^* [HMS03]	complex industrial telephony system
2004		$L_{i/o}^*$ [Nie03]	complex industrial telephony system
2004		L_{Mealy}^* [Mar+04]	complex industrial telephony system
2007		QSM [Wal+07]	Java framework example application
2010	T	Aarts et al. [AV10]	smart passport and SIP [Sch+02]
2012		RALT [SSE12]	embedded car software
2013	T B	Neider et al. [NJ13]	token ring protocol [LAN99]
2014	S	Z-Quotient [Pet+14]	1 open- and 1 closed-source SIP [Sch+02] implementation
2014		Shahbaz et al. [SG14]	Java reservation software and a home automation system
2019		∂L_M^* [DMS19]	multiple OpenSSL versions
2019	S	ADT [Fro19]	conference management software and a printer controller
2020		Duhaiby et al. [DG20]	X-ray machine
2021	T	L_{MDP}^* [Tap+21]	coin consensus protocol [AH90]
2022	B	TTT/ik [KKG22]	5 embedded system controllers
2024	T	L_{SMM}^* [Muš+24]	MQTT [OAS19], TCP [Edd22], and Bluetooth Low Energy [Cor25]
2025	B	AL [#] [KJR25]	multiple OpenSSL versions and evolutions of legacy code

OBT: other benchmark types used T: toy examples B: benchmark libraries S: synthesis

Table 4.3: Use of case studies in active FSA learning

Year	Algorithm / Authors	Case Studies	Field
2002	AMC [GPY02]	a cache coherence protocol	hybrid learning
2008	Chaki et al. [Cha+08]	IPC protocol implementation versions	correctness checking
2008	PassiveInfer [WB08]	2 Java applications	hybrid learning
2013	ACQC [Win+13]	conference management software	hybrid learning
2019	Yang et al. [Yan+19]	industrial machine controller	hybrid learning
2022	Aichernig et al. [AMP22]	Bluetooth Low Energy [Cor25] and MQTT [OAS19]	comparative study

Table 4.4: Use of case studies in other FSA research fields

Year	OBT	Algorithm / Authors	Toy Examples
1971		Hopcroft [Hop71]	bespoke
1980		Tail-Clustering [Mic80]	bespoke
1981		Modified k -Tails [Ito81]	from previous work
1984	C	Successor [RV84]	challenging tasks
1988		Partial Similarities [KS88]	challenging tasks
1991	B	FSMRED [KS91]	textbook examples
1993		DOLCE [DM93]	from previous work
1993		RPNI [OG93]	divisibility of numbers
1993		OSTIA [OGV93]	Roman and plain-English number decoding and string reversal
1993	B	Puri et al. [PG93]	textbook examples
1994	CBS	ISM [Kam+94]	challenging tasks
1996		DDDI [HOV96]	from previous work
2004	B	HSM2 [HXB04]	textbook examples
2006		DEES [HDE06]	bespoke
2008	S	MSM [LDD08]	train controller
2011		Tsarev et al. [TE11]	elevator
2018	C S	Ulyantsev et al. [UBS18]	alarm clock, elevator, and ATM
2019	C	App. IOALERGIA [AT19]	slot machine and robot navigation
2020	C	Jeppu et al. [Jep+20]	counter and integrator
2022	S	Tappler et al.	from previous work

OBT: other benchmark types used C: case studies B: benchmark libraries S: synthesis

Table 4.5: Use of toy examples in passive FSA learning

Year	OBT	Algorithm / Authors	Toy Examples
1993		Rivest-Schapire [RS93]	3 robot navigation scenarios
1994		Rivest et al. [RS94]	5 robot navigation scenarios
2008	B S	App. QSM [Dup+08]	train controller and phone
2010	C	Aarts et al. [AV10]	bespoke
2011		IKL [MS11]	cruise controller and elevator
2011		L_M^* [SHM11]	coffee machine
2012		Active IOALERGIA [CN12b]	slot machine
2012	S	L_1 [IGO12]	coffee machine
2013	C B	Neider et al. [NJ13]	modulo counter
2019	B	product- L^* [Moe19]	n -bit register machines
2021		Frohme et al. [FS21]	palindrome
2021	C	L_{MDP}^* [Tap+21]	robot mazes and a slot machine
2022		Bainczyk et al. [BSH22]	cruise controller
2024	C	L_{SMM}^* [Muš+24]	robot mazes and a slot machine

OBT: other benchmark types used C: case studies B: benchmark libraries S: synthesis

Table 4.6: Use of toy examples in active FSA learning

Year	OBT	Algorithm / Authors	Benchmark Libraries
1991		STAMINA [Hac+91]	MCNC [Yan91]
1991	T	FSMRED [KS91]	MCNC [Yan91]
1993	T	Puri et al. [PG93]	MCNC [Yan91]
1994		ALERGIA [CO94]	Reber grammar [Reb67]
1994		App. OSTIA [CGV94]	Miniature Learning Acquisition [Fel+90]
1994	C T S	ISM [Kam+94]	MCNC [Yan91]
1995	C	Ron [Ron95]	TIMIT [ZSG90]
1995		Sánchez et al. [SGL95]	MCNC [Yan91]
1996		SLIM [HM96]	ISM [Kam+94] and MCNC [Yan91]
1998	C	Markov [CW98]	IPSW 6/7 process problem [Kel+90]
1998		EDSM [LPP98]	Abbadingo One [LPP98]
1998		DD-OSTIA [Onc98]	Miniature Learning Acquisition [Fel+90]
1999		exbar, ed-beam [Lan99]	extended BIC [OS98]
1999		bica [PO99]	ISM [Kam+94]
2000		MDI [TDH00]	ATIS corpus [Hir+93]
2001		VOID [AD01]	MCNC [Yan91]
2002		(Blue-)Malergia [KD02]	Reber grammar [Reb67]
2004	T	HSM2 [HXB04]	MCNC [Yan91]
2006		MDL [AJ06]	Abbadingo One [LPP98]
2007		chesmin [GF07]	ISM [Kam+94]
2010		Heule et al. [HV10]	extended BIC [OS98]
2013		cosme [AS13]	ISM [Kam+94]
2013		MuACOSm [CU13]	artificial ant problem instances [Jef+92]
2013		DFASAT [HV13]	StaMinA [Wal+13]
2013		Klimowicz et al. [KS13]	MCNC [Yan91]
2015		MeMin [AR15]	ISM [Kam+94] and MCNC [Yan91]
2019	S	Petrenko et al. [Pet+19]	Automata Wiki [Nei+19]
2021		DISC [Shv+21]	ALFRED [Shr+20], Crystal Island [Ha+11], malware [Ber+19], MIT-AR [TIL04], StarCraft [KOG19]
2026		Lutz et al. [Lut+26]	ALFRED [Shr+20], HDfS [Xu+09], Blue Gene/L [OS07]

OBT: other benchmark types used C: case studies T: toy examples S: synthesis

Table 4.7: Use of benchmark libraries in passive FSA learning

Year	OBT	Algorithm / Authors	Benchmark Libraries
2008	B S	App. QSM [Dup+08]	mine pump specification [Jos01]
2009	S	L_M^+ [SG09]	Edinburgh Concurrency Workbench [MS99]
2012		Observation Pack [How12]	Edinburgh Concurrency Workbench [MS99] and CADP [Gar+13]
2013	C T	Neider et al. [NJ13]	FASTer [BFL04] and T(O)RMC [Leg08] benchmarks
2014		TTT [IHS14]	Edinburgh Concurrency Workbench [MS99]
2019	T	product- L^* [Moe19]	LGSynth'91 [Yan91]
2022		L^λ [HS22]	Automata Wiki [Nei+19]
2022	C	TTT/ik [KKG22]	Automata Wiki [Nei+19]
2022		$L^\#$ [Vaa+22]	Automata Wiki [Nei+19]
2023		CL^* [Lab+23]	Body Comfort System case study [Lit+17]
2024		Active RPNI [ATW24]	Automata Wiki [Nei+19]
2024		Baumgartner et al. [BV24]	CTU-13 [Gar+14], HDFs [Xu+09], and Blue Gene/L [OS07]
2025	S	Henry et al. [Hen+25]	2019 Model Checking Contest [Amp+19], Zuberek's Petri net library [Zub99], and the Hippo-CPS collection [Wiś+23]
2025	C	$AL^\#$ [KJR25]	Automata Wiki [Nei+19]
2025	S	SCL^* [SBH25]	Body Comfort System case study [Lit+17]
2025	S	$L^\#$ for DFAs [VS25]	Automata Wiki [Nei+19]

OBT: other benchmark types used C: case studies T: toy examples S: synthesis

Table 4.8: Use of benchmark libraries in active FSA learning

Year	OBT	Algorithm / Authors	Benchmark Libraries	Field
2005		Damas et al. [Dam+05]	mine pump specification [Jos01]	hybrid learning
2015	S	Hierons et al. [HC15]	LGSynth'91 [Yan91]	distinguishing seq.
2024		CASTLE [Tap+24]	OpenAI Gym [Bro+16]	hybrid learning

OBT: other benchmark types used S: synthesis

Table 4.9: Use of benchmark libraries in other FSA research fields

Year	OBT	Algorithm / Authors	Cat.	Synthesis Approach
1994	C T B	ISM [Kam+94]	Str.	random automata
1997		sk-strings [RP97]	Str.	random automata
1998		BIC [OS98]	Str.	random automata
2002		W-EDSM [CK02]	Str.	Abbadingo One [LPP98]
2004		DeLeTe2 [DLT04]	Both	random DFAs, random NFAs, and random regular expressions
2005		Smart [LR05]	Str.	Abbadingo One [LPP98]
2008	T	MSM [LDD08]	Str.	random automata
2015		DFA-Inductor [UZS15]	Str.	random automata
2018	C	Avellaneda et al. [AP18]	Str.	random automata
2018	C T	Ulyantsev et al. [UBS18]	Str.	random automata
2018		DFA-Inductor [ZSU18]	Str.	random automata
2019	B	Petrenko et al. [Pet+19]	Str.	random automata
2019		DFA-Inductor 2 [Zak+19]	Str.	random automata
2022	T	Tappler et al. [TAL22]	Str.	random automata

OBT: other benchmark types used C: case studies T: toy examples B: benchmark libraries
Cat.: Synthesis Category Str.: structural

Table 4.10: Use of synthetic benchmarks in passive FSA learning

Year	OBT	Algorithm / Authors	Cat.	Synthesis Approach
2008	T B	App. QSM [Dup+08]	Str.	modified Abbadingo One [LPP98]
2009		NL* [Bol+09]	Both	random DFAs, random NFAs, and random regular expressions
2009	B	L_M^+ [SG09]	Str.	random automata
2012	T	L_1 [IGO12]	Str.	random automata
2014	C	Z-Quotient [Pet+14]	Str.	random automata
2015		NL*, UL*, AL* [AEF15]	Str.	random automata
2018		Groz et al. [Gro+18]	Str.	random automata
2019	C	ADT [Fro19]	Str.	random automata with optional added loops
2022		Ferreira et al. [FHS22]	Str.	random automata
2025	B	Henry et al. [Hen+25]	Str.	random automata
2025	B	SCL* [SBH25]	Str.	reflects real network topologies
2025	B	$L^\#$ for DFAs [VS25]	Str.	random automata

OBT: other benchmark types used C: case studies T: toy examples B: benchmark libraries

Cat.: Synthesis Category Str.: structural

Table 4.11: Use of synthetic benchmarks in active FSA learning

Year	OBT	Algorithm / Authors	Cat.	Synthesis Approach
2009		Hierons et al. [Hie+09]	Str.	strongly connected automata
2012		Güniçen et al. [Gün+12]	Str.	strongly connected automata
2015	B	Hierons et al. [HC15]	Str.	selected random automata

OBT: other benchmark types used B: benchmark libraries **Cat.**: Synthesis Category Str.: structural

Table 4.12: Use of synthetic benchmarks in FSA distinguishing sequence generation

[IHS14] on the toy examples used by the authors, but not (consistently) on the benchmark library used in conjunction with them. In a more unusual case, Neider and Jansen [NJ13] needed to augment their benchmark library to show scaling effects using additional toy examples. Tables 4.5 and 4.6 list all works in this category.

Benchmark libraries are a staple of passive learning research since 1990 and active learning since ca. 2010. Especially in passive learning, they are often used standalone (i.e., without other categories of benchmark). When correlating libraries and publications' dates, it becomes apparent that benchmark libraries seem to be eventually replaced with newer, more complex alternatives. In passive learning, MCNC [Yan91] was last used in 2013, while newer works use larger data sets such as ALFRED [Shr+20]. A list of all works is shown in Tables 4.7 to 4.9. $L^\#$ for DFAs [VS25] is of particular interest, since the algorithm works well on synthetic automata, but the addition of a library showed severe scaling problems not apparent from the synthetic automata.

Finally, the use of synthesized automata is becoming increasingly common in recent years. Almost all works employ structural synthesis by simple random generation (i.e., generate a number of states, randomly assign transitions, etc.). Occasionally, established generation algorithms are used. DeLeTe2 [DLT04] and NL* [Bol+09] are exceptions and use both random generation and semantic synthesis. Their method generates an automaton's language via random regular expressions. Synthesis is mostly used in conjunction with other approaches, likely due to the well-known downsides of random generation such as limited language complexity. Tables 4.10 to 4.12 show an overview of all works in this category.

In all categories, combining multiple benchmark categories is becoming increasingly common. Most works published since 2020 use two or more categories of benchmark in their evaluation.

A visualization of these trends can be seen in Figure 4.1. Here, the chronological progressions described above are easily visible. Toy examples replace theoretical works and are then replaced or augmented by case studies and benchmark libraries. Synthetic benchmarks are a more recent trend, but still less common, as is the use of multiple benchmark categories. Since FSA research is a more mature field and – as opposed to EFSA research – has mostly settled on a fixed set of automaton models, we expect these observations to be typical of a well-developed research field.

4.1.2 Extended Finite State Automaton Research

Research into EFSA from an algorithmic perspective with a focus on learning and model checking began in the 2000s, so we expect a less developed field w.r.t. benchmarking. Additionally, EFSA are not interchangeable based on the model: for example, timed automata and register automata are fundamentally different and the results shown in Chapter 3 demonstrate the limits of transformations even between RA formalisms. We retain the categorization used for FSA research in the previous section.

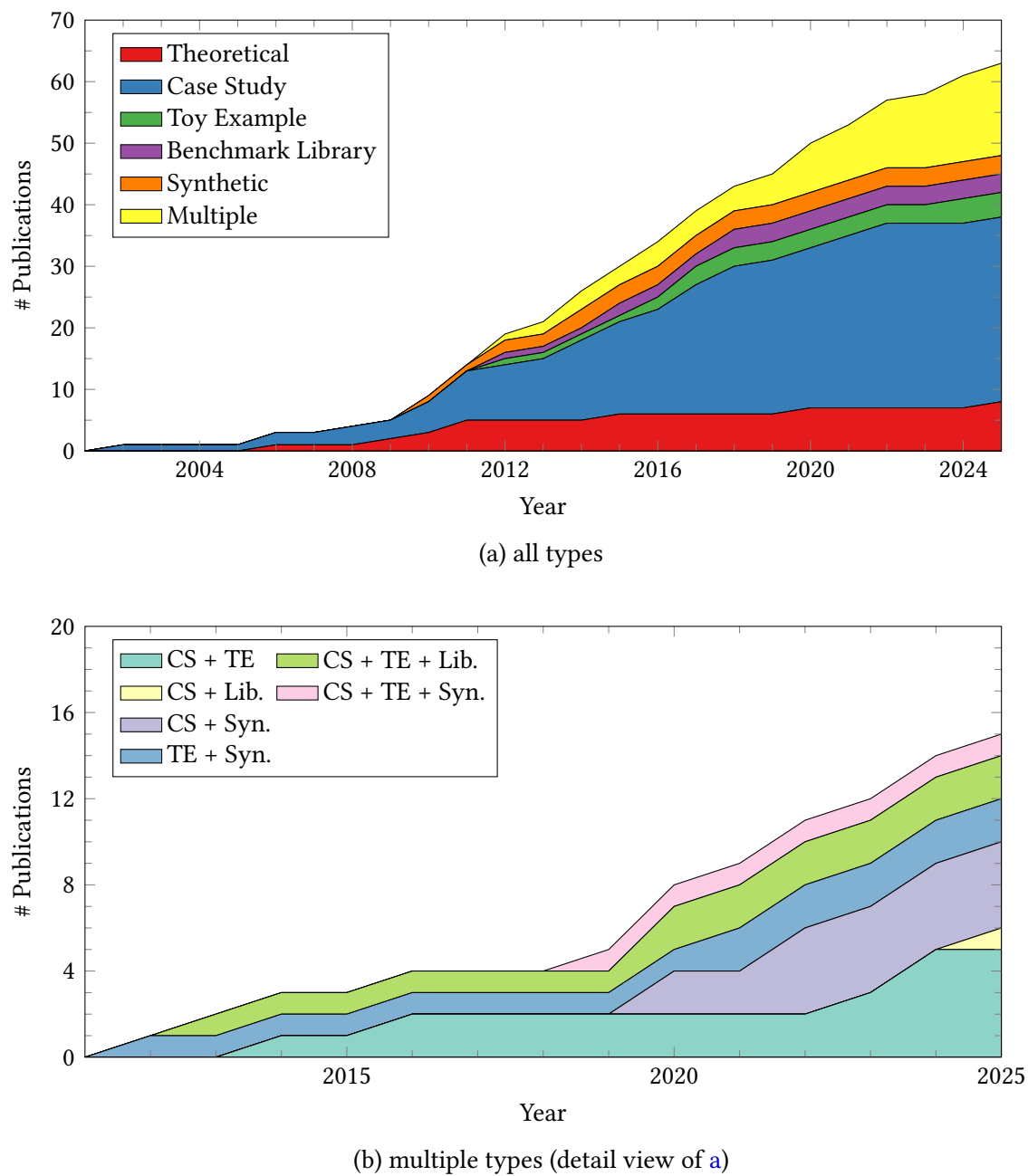


Figure 4.2: Evolution of benchmark types in EFSA research

Year	Algorithm / Authors	Year	Algorithm / Authors
2006	Grinchtein et al. [GJP06]	2009	Demri et al. [DL09]
2010	TL_{sg}^* , TL_{nsg}^* , TL_s^* [GJL10]		
2011	TL^* [Lin+11]		
2011	ID_1-DTA [VWW11]		
2015	Mens et al. [MM15]		
2020	Henry et al. [HJM20]		
2025	L_α [Fra+25]		

(a) active automata learning

(b) model checking

Table 4.13: Purely theoretical works in EFSA research by field

Year	OBT	Algorithm / Authors	Case Studies
2002		Missing-Data EM [WWT02]	TCP [Edd22] and HTTP [TB22] networking
2008		GK-tail [LMP08]	5 Java applications
2011		Synoptic [Bes+11]	reverse trace-route and a university course
2011		jMINER [LCR11]	4 Java SE packages
2011		BUTLA [Mai+11]	miniature smart factory
2012		Matos Pedro et al. [MCS12]	multi-processor scheduler
2014		OTALA [Mai14]	miniature smart factory
2014	T	Perfume [Ohm+14]	RADIUS [Rub+97], browser caching, TCP [Edd22], and web application usage
2016		MINTm [WTD16]	2 Erlang applications and 3 Java SE classes
2017		GK-tail+ [MPS17]	3 Java libraries
2017		eRTI+ [WKK17]	ATM logs
2018		ReHMM [EM18]	3 Erlang applications and 4 Java SE classes
2019	T S	Tappler et al. [Tap+19]	car alarm and particle counter
2022	S	TAG [Cor+22]	TV broadcast information
2022		TkT [Pas+22]	4 algorithm implementations and 3 Java libraries

OBT: other benchmark types used T: toy examples S: synthesis

Table 4.14: Use of case studies in passive EFSA learning

Year	OBT	Algorithm / Authors	Case Studies
2006		PSYS [BJR06]	proprietary network protocol
2010		Aarts et al. [AJU10]	TCP [Edd22] and SIP [Sch+02]
2010		App. Aarts et al. [ASV10]	biometric passport
2013		App. Aarts et al. [ADP13]	bank card
2013	T B	Isberner et al. [IHS13]	biometric passport
2014		App. Aarts et al. [Cha+14]	card reader
2014		App. Aarts et al. [FJV14]	TCP [Edd22] stacks
2015		App. Aarts et al. [RP15]	TLS [Res18] stacks
2015		App. Aarts et al. [Sme+15]	printer controller
2016	T	SL* [Cas+16]	1 Java SE data structure and TCP [Edd22]
2016		App. Aarts et al. [FJV16]	TCP [Edd22] stacks
2017		App. Aarts et al. [Fit+17]	SSH [LY06] implementations
2017		App. Aarts et al. [TAB17]	MQTT [OAS19] implementations
2018		App. Aarts et al. [DPR18]	OpenVPN
2018		App. various [MCR18]	802.11 handshakes [LAN25]
2019		App. Aarts et al. [RAR19]	QUIC [IT21] implementations
2020	S	DOTAL [An+20]	TCP [Edd22]
2020		App. Aarts et al. [Fit+20]	DTLS [RTM22] stacks
2020	T B	Garhewal et al. [Gar+20]	2 Python data structures
2021		App. Aarts et al. [PA21]	Bluetooth [Cor25] devices
2021		Vaandrager et al. [VBE21]	car alarm, particle counter, TCP [Edd22], and Android WiFi key management
2022		App. Aarts et al. [PA22]	Bluetooth [Cor25] devices
2022	S	SL [XAZ22]	car alarm, particle counter, TCP [Edd22], and Android WiFi key management
2023	T	MMLT [KKG23]	car alarm, particle counter, TCP [Edd22], SCTP [STN22], and Android WiFi key management
2024	T	MMLT/ik [KSG24]	car alarm, particle counter, TCP [Edd22], SCTP [STN22], and Android WiFi key management
2025	B	Dengler et al. [DAH25]	PostgreSQL

OBT: other benchmark types used T: toy examples B: benchmark libraries S: synthesis

Table 4.15: Use of case studies in active EFSA learning

Year	OBT	Algorithm / Authors	Case Studies	Field
2020	S	Aichernig et al. [APT20]	smart light switch, car alarm, and a particle counter	comparative study
2024	T	Klein et al. [KKG24a]	various industrial software, TCP [Edd22], SCTP [STN22], and Android WiFi key management	state estimation
OBT: other benchmark types used T: toy examples S: synthesis				

Table 4.16: Use of case studies in other EFSA research fields

As was the case with FSAs, purely theoretical works are only common in seminal works (e.g., TL^* [Lin+11]) and papers describing a theory-centric contribution (e.g., Demri and Lazić [DL09]). However, they are less common; we are not aware of any such work in passive EFSA learning. Table 4.13 shows an overview of all works in this group.

Case studies are an extremely common benchmark category in EFSA research, with roughly half the works in this survey being pure case studies. This is primarily due to the large number of publications in which the abstraction-based learner pattern by Aarts et al. [AJU10] is applied to learn models of various systems, mostly network protocols, in order to perform security-related analyses. Recent works in this category tend to employ other benchmark categories in conjunction with case studies. Tables 4.14 to 4.16 list all works in this category.

Toy examples are far less common in EFSA research and are almost never used standalone. This indicates that the use of toy examples in FSA research may be typical for an intermediate time period before case studies were commonly available around 2000. Since EFSA research started only then, toy examples were less needed. All toy examples are clearly real-world inspired (e.g., an oven or an alarm clock) and are often reused. They usually act to supplement case studies with simpler examples in an evaluation. Table 4.17 lists all works using toy examples.

The use of benchmark libraries in EFSA research is extremely uncommon, with all but one work using the Automata Wiki [NJ13]. The remaining work uses benchmark libraries that do not contain extended FSAs to establish a performance baseline. A list of all relevant papers is shown in Table 4.18.

Synthesis is more common than the use of benchmark libraries, but also rather limited and usually used in conjunction with other approaches. With the exception of NRTALearning [An+21], which also uses random regular expressions (i.e., semantic synthesis), only random structural synthesis was used. Table 4.19 names all works using synthesis.

Except for a large set of standalone case studies, the use of multiple benchmarks is relatively standard in EFSA research. Especially recent works almost uniformly use multiple approaches.

Year	OBT	Algorithm / Authors	Toy Examples
2012	S	Ulyantsev et al. [UT12]	alarm clock
2014	C	Perfume [Ohm+14]	network connection tester
2016		IOALERGIA [Mao+16]	slot machine and a workstation cluster
2019	C S	Tappler et al. [Tap+19]	smart light switch and a train controller
OBT: other benchmark types used C: case studies S: synthesis			
(a) passive automata learning			

Year	OBT	Algorithm / Authors	Toy Examples
2012		L_{RMM}^* [How+12a]	3 data structures
2013	C B	Isberner et al. [IHS13]	calling code, file system navigation, and bespoke
2016	C	SL^* [Cas+16]	alternating bit communication, 2 number sequences, timeouts, and a prepaid card
2017		vL^* , vNL^* [Moe+17]	queue and bespoke
2020	C B	Garhewal et al. [Gar+20]	combination lock and bespoke
2021	S	NRTALearning [An+21]	machine scheduling
2023	C	MMLT [KKG23]	wireless device and oven
2024	C	MMLT/ik [KSG24]	wireless device, oven, HVAC controller, and washing machine
OBT: other benchmark types used C: case studies B: benchmark libraries S: synthesis			
(b) active automata learning			

Year	OBT	Algorithm / Authors	Toy Examples	Field
2024	C	Klein et al. [KKG24a]	test scheduling, oven, and a HVAC controller	state estimation
2024		Klein et al. [KKG24b]	cloud system, medical software, sensor mesh, and an ATM	opacity verif.
OBT: other benchmark types used C: case studies				
(c) other fields				

Table 4.17: Use of toy examples in EFSA research by field

Year	Algorithm / Authors	Benchmark Libraries
2018	Smetsers et al. [SFV18]	Automata Wiki [Nei+19]

(a) passive automata learning

Year	OBT	Algorithm / Authors	Benchmark Libraries
2012		Tomte [Aar+12]	predecessor of Automata Wiki [Nei+19]
2013	C T	Isberner et al. [IHS13]	Edinburgh Concurrency Workbench [MS99] and CADP [Gar+13]
2015		Tomte [Aar+15]	predecessor of Automata Wiki [Nei+19]
2020	C T	Garhewal et al. [Gar+20]	Automata Wiki [Nei+19]
2025	C	Dengler et al. [DAH25]	Automata Wiki [Nei+19]

OBT: other benchmark types used C: case studies T: toy examples

(b) active automata learning

Table 4.18: Use of benchmark libraries in EFSA research by field

Year	OBT	Algorithm / Authors	Cat.	Synthesis Approach
2010		RTI+ [VWW10]	Str.	random automata
2012	T	Ulyantsev et al. [UT12]	Str.	random automata
2012		RTI [VWW12]	Str.	random automata
2014		CSP+MuACOSm [CUS14]	Str.	random automata
2019	C T	Tappler et al. [Tap+19]	Str.	random automata
2022	C	TAG [Cor+22]	Str.	random automata
OBT: other benchmark types used C: case studies T: toy examples Cat.: Synthesis Category Str.: structural				

(a) passive automata learning

Year	OBT	Algorithm / Authors	Cat.	Synthesis Approach
2020	C	DOTAL [An+20]	Str.	random automata
2021	T	NRTALearning [An+21]	Both	random NRTAs and random regular expressions
2022	C	SL [XAZ22]	Str.	random automata
OBT: other benchmark types used C: case studies T: toy examples Cat.: Synthesis Category Str.: structural				

(b) active automata learning

Year	OBT	Algorithm / Authors	Cat.	Synthesis Approach
2020	C	Aichernig et al. [APT20]	Str.	random automata
OBT: other benchmark types used C: case studies Cat.: Synthesis Category Str.: structural				

(c) comparative studies

Table 4.19: Use of synthetic benchmarks in EFSA research by field

The results are shown in [Figure 4.2](#) (using the same grouping as previously). As opposed to FSA research, we see no “toy example phase” prior to adoption of case studies; toy examples are rarely used at all. The large number of case studies is due to the large number of works applying abstraction-based learning Aarts et al. [[AJU10](#)]. Use of benchmark libraries is extremely rare, and library-synthesis combinations were never used in the surveyed body of work.

4.2 Case Study I: REACH via Backwards Propagation

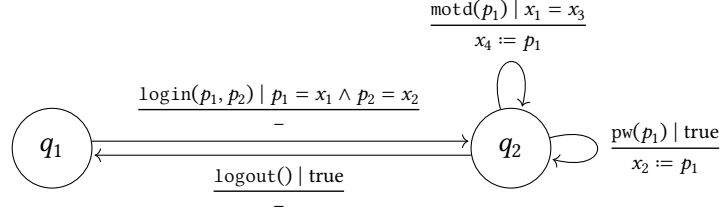
The first case study considered here concerns model checking of register automata, i.e., proving or falsifying statements about a given register automaton. The most basic question for register automata is that of *reachability*, i.e., if a given location in an RA can be reached via an input. Reachability testing can be used to answer further questions, e.g., non-emptiness (can an accepting state be reached) or more complex model checking question via the construction of product automata.

Our algorithm introduces a transformation into *history independent* register automata, which preserves an RAs semantics while ensuring that reachability can be trivially decided. We then evaluated this algorithm on a benchmark library and structurally synthesized automata to consider its performance on average automata as opposed to the worst case. In [Section 4.2.1](#), we summarize the algorithmic approach, in [Section 4.2.2](#), analyze the choice of benchmark approach, and in [Section 4.2.3](#), recap the evaluation results and review the impact of benchmark selection w.r.t. the study results’ empirical content.

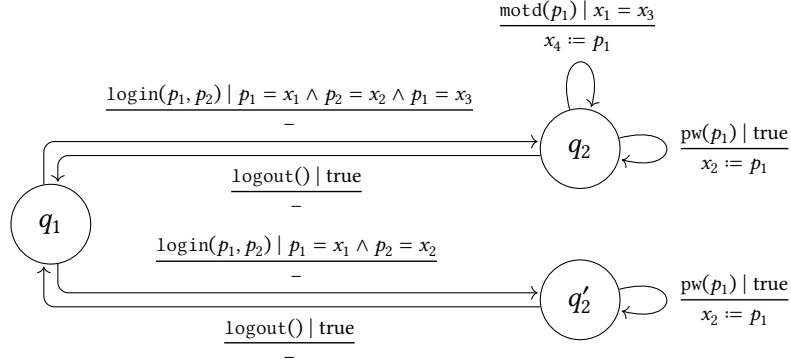
This section is based on [Paper II \[DH22a\]](#), but considers its results from a benchmarking perspective, focusing on benchmark selection and its impacts. We also provide revised evaluation results that address an error in the original implementation which degraded the apparent performance of our approach.

4.2.1 Background and Approach

Our approach is motivated by Murawski et al.’s *register disciplines* [[MRT17](#)]: the source of complexity for deciding reachability is the amount of “hidden” state that is stored in the automaton’s registers and can be used to reject transitions. This includes initializing registers to # (if a guard demands an equal value, it cannot be satisfied) and, more critically, using equality relations between registers in a guard (if they are not, guard is effectively disabled). However, storing information and checking it “after the fact” is an unusual pattern: e.g., the XMPP automaton in [Figure 1.1](#) does not exhibit it. Our approach transforms automata into a semantically equivalent representation in which this pattern does not occur.



(a) history dependent



(b) history independent

Figure 4.3: Modified XMPP operations with and without history independence

First, we formalize a property of register automata in which no such “after-the-fact” checking happens. This is closely related to the notion of *right-invariance* [Cas+15].

Definition 5 (History Independence). Given a register automaton (Q, X, s_0, Q^+, Γ) over U , we define the set of reachable states for a location $q \in Q$ by

$$S_q = \{\langle q, \chi \rangle \mid \exists w \in U^* : \langle q, \chi \rangle \in \delta(s_0, w)\}.$$

A transition $\langle q, \lambda, g, u, q' \rangle$ is *history independent* if for all reachable states $\langle q, \chi \rangle \in S_q$, there exists at least one data symbol $\lambda(d_1, \dots, d_{a(\lambda)}) \in U$ such that the combined current valuation $v \models g$, i.e., the transition’s guard can be satisfied.

If all transitions in Γ are history independent, we call the register automaton *history independent*.

An example of this can be seen in the modified XMPP example in Figure 4.3 that adds an administrative operation: if the user is an administrator (i.e., the name matches the one stored in x_3), they may set the message of the day. In Figure 4.3a, this operation is not implemented

history-independently, since we can reach states in q_2 in which the `motd` transition can not be taken for any input. The version in [Figure 4.3b](#), on the other hand, is history independent.

In a history independent RA, for each path through the automaton, a matching sequence of input symbols can be constructed. As a result, reachability of location follows from reachability in the graph structure and can be checked in NL. However, since the reachability of states in succinct RAs is a PSPACE-complete problem [DL09], any transformation must be FSPACE-complete, i.e., not efficiently computable in all cases under common complexity assumptions.

In this case study, we developed an algorithm operating on succinct RAs that converts an RA into a semantically identical, history independent version. We will only discuss the intuition behind the method and refer to [Paper II \[DH22a\]](#) for the detailed description and correctness proofs.

Conceptually, we track invariants for each location that define relations between registers. For example, in [Figure 4.3b](#), state q'_2 maintains the invariant $x_1 = x_3$. The algorithm operates incrementally. During each step, we ensure that each location claims a sufficient invariant to a) ensure that each outgoing transition is available and b) afterwards, each successor location's invariant is maintained. For each location, we compute the necessary invariants for each outgoing transition and split the locations into one version per invariant (plus one “terminal” version without invariants). The process is repeated until no new locations are created; we then discard all locations that have a contradictory invariant (i.e., $x_i \neq x_i$) or become unreachable. The resulting automaton is history independent and retains the original's behavior.

4.2.2 Benchmark Selection

A theoretical analysis shows that our algorithm will perform badly in *some* cases. The driving question of the evaluation therefore is the algorithm's performance on real-world automata and its scaling behavior to estimate how “common” automata that do require a large amount of new locations are. For example, the XMPP automaton in [Figure 1.1](#) is already history independent and would remain untouched by the algorithm.

Since this work was not motivated by a case study, we did not use one for evaluation. While toy examples were used during the development and implementation of the algorithm for unit testing, they are not suitable for answering the evaluation question, since they are neither representative of the real world nor sufficiently large to analyze scaling effects.

We therefore conducted a first round of benchmarking on the only suitable library of succinct RA benchmarks, the Automata Wiki [Nei+19]. It contains mostly real world or real world-inspired automata, answering the question if such automata can be efficiently transformed.

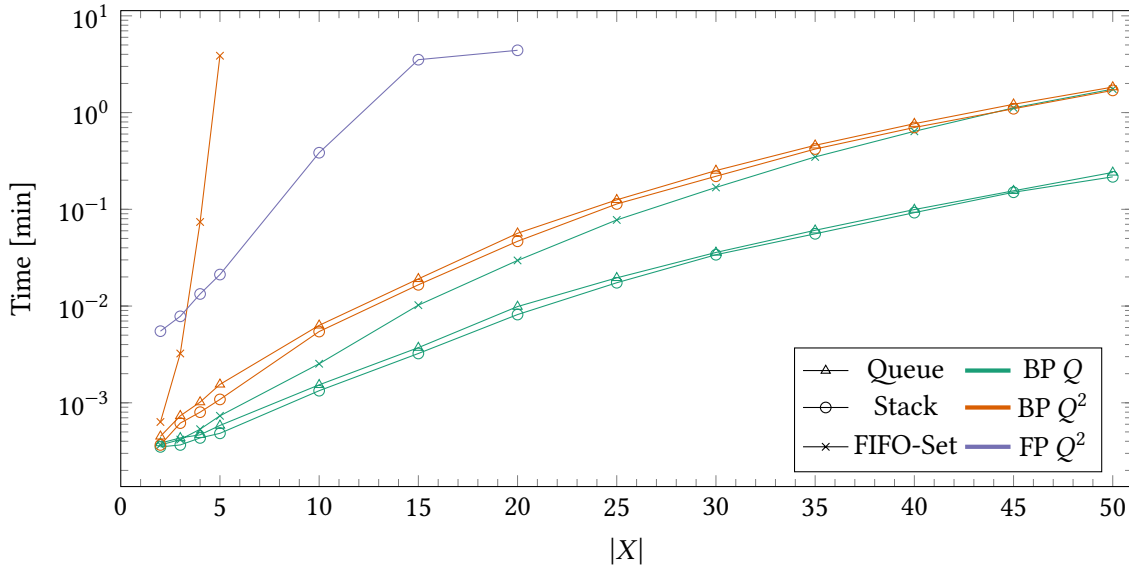


Figure 4.4: Backwards vs. forwards propagation performance on Automata Wiki data structure automata

To analyze scaling effects, we turned to synthesized automata. Since the algorithm operates on the automaton structure, we developed the structural synthesis method described in Section 5.1. This method guarantees that guards do not contain direct contradictions and can scale most properties, including size, number of registers, and guard complexity. In the evaluation, we analyzed scaling effects on the former two.

4.2.3 Evaluation Results

In the evaluation, we used three analysis scenarios: direct application of backwards propagation to the automaton (BP Q), backwards propagation on the cross product of the automaton with itself (BP Q^2 , simulating a more complex model checking scenario), and, as a baseline, the forwards propagating equivalence checker bundled with RALib⁴ [CFJ15] that uses SMT solving. This checker was also used to check the equivalence of the automaton with itself.

The Automata Wiki library contains two classes of RA models. First, it contains families of models of data structures, namely stacks, queues, and FIFO-sets (queues with a uniqueness guarantee) with increasing maximum storage (corresponding to the number of registers). The evaluation results on these are shown in Figure 4.4. Since these automata are history-independent by design, our algorithms only need to prove this fact. They were able to do substantially faster than the forwards-propagating analyzer, which timed out on all analyses but

⁴<https://github.com/LearnLib/ralib>

Automaton			BP Q			BP Q^2			FP Q^2
Name	$ Q $	$ \Gamma $	Q -BU	Γ -BU	Time	Q -BU	Γ -BU	Time	Time
ABP Channel	5	8	200 %	188 %	0.04 s	143 %	150 %	0.13 s	DNF
ABP Output	30	50	100 %	100 %	0.03 s	80 %	92 %	0.12 s	DNF
ABP Receiver 3	6	13	383 %	292 %	0.04 s	1 083 %	426 %	0.90 s	DNF
FWGC	18	43	289 %	179 %	0.09 s	782 %	374 %	1 098.30 s	0.40 s
Login	12	20	125 %	115 %	0.03 s	250 %	223 %	0.07 s	0.29 s
Map	17	28	135 %	136 %	0.04 s	98 %	100 %	0.49 s	0.57 s
Overwriting Map	15	24	153 %	158 %	0.03 s	77 %	88 %	0.24 s	0.65 s
Passport	35	82	129 %	112 %	0.07 s	157 %	122 %	0.26 s	0.71 s
Repdigit Palindrome	6	23	400 %	178 %	0.04 s	1 033 %	192 %	0.12 s	0.31 s
SIP	27	65	100 %	100 %	0.03 s	113 %	93 %	0.07 s	25.90 s

BP: Backwards Propagation BU: Blowup FP: Forwards Propagation DNF: Execution failed with an error

Table 4.20: Backwards vs. forwards propagation performance on non-family Automata Wiki automata

stacks up to size 20. Self-products of FIFO-sets form an exception: here, we could observe blowups and backwards-propagating analyses timed out for sizes 10 and more.

Second, the Wiki contains a set of (non-family) automata mostly derived from real-world scenarios. The results on these are shown in Table 4.20. In three cases, the RALib-based tool failed to load the automaton. In all but two of the remaining scenarios, both approaches required less than one second of wall clock time, however, our approach was slightly faster. In the farmer-wolf-goat-cabbage puzzle⁵ (FWGC), backwards propagation is substantially slower, while on the SIP [Sch+02] model, forwards propagation is slower.

Finally, we applied our approach to synthesized automata. Since they do not follow the input-output alternation required by the RALib model checker, we only performed simple transformations on them and did not create self-products. The results are shown in Figure 4.5 and mostly confirmed our expectations w.r.t. scaling. When scaling up the number of registers (cf. Figure 4.5a), analysis time grows polynomially and the blowup increases linearly, which seems tolerable in practical use. When fixing the number of states and adding registers, this does not happen (cf. Figure 4.5a); blowup remains constant and analysis time scales linearly.

We can conclude that the backwards-propagating approach is superior in almost all observed scenarios. Since the Automata Wiki systems are derived from real-world systems, we can expect this to hold true for unknown systems, as well. Using synthetic automata, we can show

⁵A classic puzzle also known as, e.g., the river crossing puzzle [PS89].

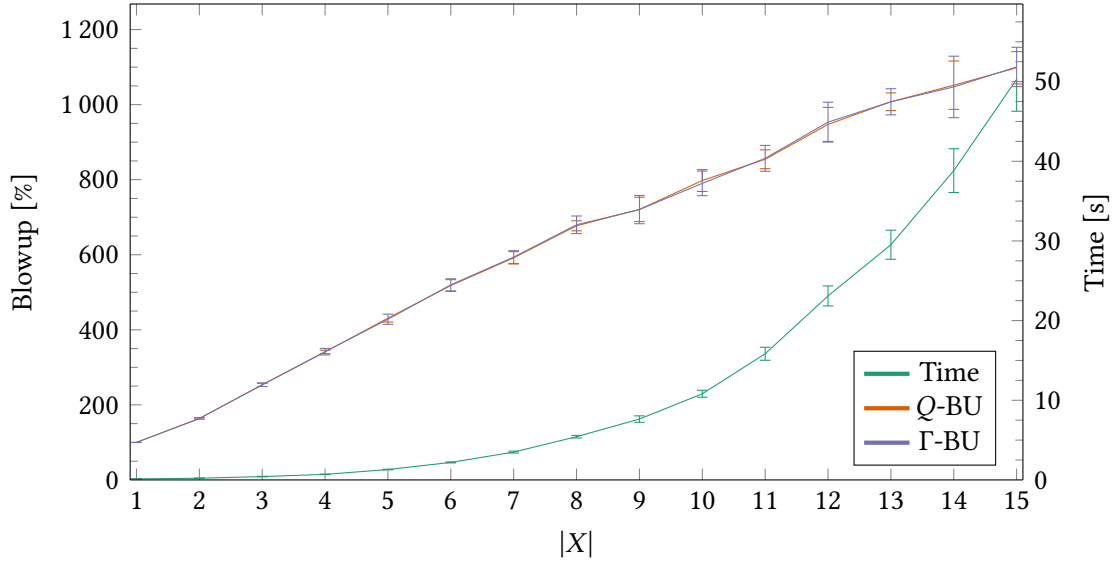
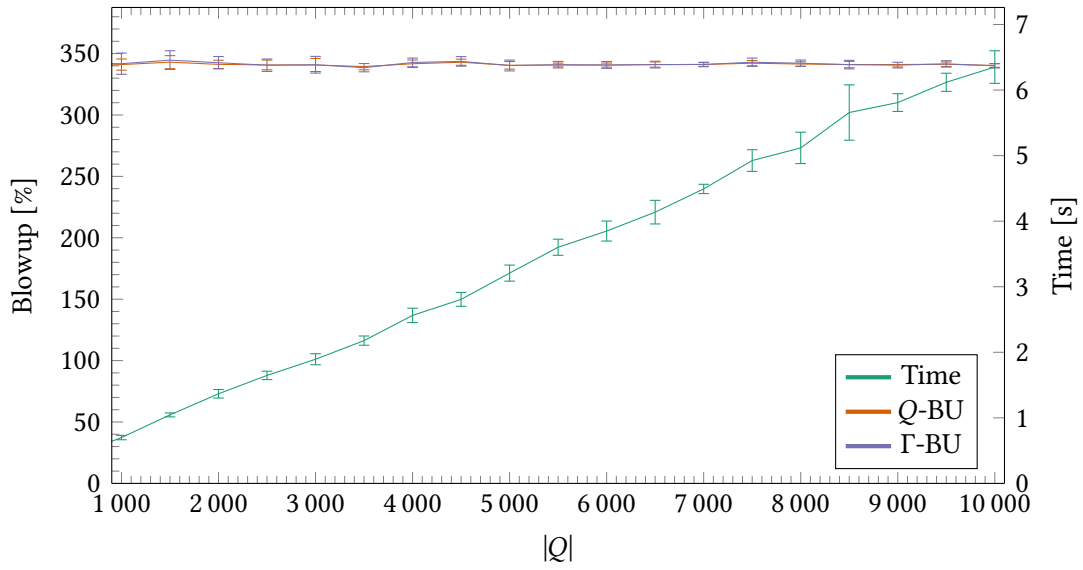
(a) $|Q| = 1000$, dynamic $|X|$ (b) dynamic $|Q|$, $|X| = 4$

Figure 4.5: Backwards propagation performance on structurally synthesized automata

that our algorithm can handle even very large systems. This conclusion can not be made using the Wiki automata, since the scaling examples do not exhibit any history dependence, i.e., are outliers in favor of our approach.

In summary, the benchmarking methods worked complementarily, each supporting a different aspect of the evaluation’s empirical content. While the Wiki automata enable comparisons to the baseline and predictions about the behavior on automata encountered in the future, the synthesized ones enable predictions w.r.t. scaling effects and behavior on large models.

4.3 Case Study II: SL^λ Learning Algorithm

The second case study we examine concerns active register automata learning, i.e., deriving an automaton from an oracle that answers queries about membership (“is input w accepted?”) and equivalence (“is $RA \ A$ equivalent to the ground truth?”). The second oracle can be implemented in terms of the former by comparing the results of random walks. This method can be applied to derive a model of a black-box system that can then be reviewed by a human or subjected to model checking. The best known algorithm for this is SL^* [Cas+16], which operates similar to the L^* algorithm [Ang87]. As a result, it does not implement several active learning optimizations that are present in the L^λ algorithm [HS22].

This case study combines the two lines of work into the SL^λ algorithm, which enhances SL^* with the optimizations from L^λ plus an additional optimization. To compare its performance to the SL^* baseline, we evaluated it on a case study, a benchmark library and structurally synthesized automata. We also measure the individual optimizations’ contribution to the result. Similar to the [previous section](#), in [Section 4.3.1](#), we will sketch the optimizations’ central ideas, in [Section 4.3.2](#), we will analyze the choice of benchmark approach, and in [Section 4.3.3](#), we will summarize the evaluation results and review the impact of benchmark selection w.r.t. the study’s empirical content.

This section is based on [Paper IV](#) [Die+24; Die+26a], but considers its results from a benchmarking perspective, focusing on benchmark selection and its impacts.

4.3.1 Background and Approach

In active learning of register automata, the SL^* algorithm [Cas+16] is the state of the art, being the first active algorithm that learns register automata without abstraction.⁶ It is a relatively straightforward port of L^* [Ang87] to a symbolic setting, the main change being the introduction of *symbolic tree queries* as an intermediate query formalism.

⁶While Aarts et al.’s earlier method can learn models of such systems, it yields a Mealy machine plus a abstraction layer that reduces infinite alphabets to finite symbol sets [AJU10].

In FSA learning, several optimizations that result in an immense increase in performance have been developed, with L^λ [HS22] combining most of these optimizations. SL^λ “ports” L^λ for register automata learning, combining symbolic tree queries with the learner architecture and optimizations of L^λ . Additionally, SL^λ adds the novel restricted suffix optimization. We will give an intuition for all three optimizations, but refer to Paper IV [Die+24; Die+26a] for the detailed algorithms and proofs.

Learning algorithms using the minimum adequate teacher framework pioneered by Angluin for L^* [Ang87] operate by building a set of *prefixes* and a set of *suffixes* corresponding to the algorithm’s current hypothesis of the “correct” automaton. Prefixes correspond to locations of the automaton and define an input word that can be used to reach the locations. Ignoring parameters, in the XMPP example (cf. Figure 1.1), q_0 could be identified by the empty word ε , while q_1 could be reached by `register`. Suffixes help distinguish locations by acceptance behavior: e.g., the word `login` leads to an accepting location starting in q_1 , but not q_0 . In the case of register automate, we maintain *symbolic suffixes* that contain variables instead of data values. In general, the algorithm’s performance hinges on keeping both prefixes and suffixes as short as possible.

The short prefix optimization designates a *representative prefix* for each hypothesized location. For example, in the XMPP automaton⁷ both the prefixes

`register(user, pass), login(user, pass), logout(), register(user, pass)`

and `register(user, pass)` refer to the same location. Using the latter is preferable, since it shortens queries. The data structure responsible for maintaining this information is a *classification tree*, a technique established by Kearns and Vazirani [KV94].

A short suffix optimization was already performed by L^* [Ang87], however, we need to adapt it by analyzing counterexamples obtained in response to incorrect hypotheses. A counterexample is split into a prefix and suffix. We then find the longest prefix such that the suffix (which is then the shortest suffix) is still a counterexample and only include that into the tree. The prefix will then be replaced by the representative prefix of its location in the future.

The third optimization is the *restrictive suffix* optimization and is specific to symbolic suffixes. For example, the suffix `register( $p_1, p_1$ ), login( $p_1, p_1$ )` would suffice to differentiate q_0 from other states. However, it omits the information that the username and password parameters are independent. We therefore add additional information to suffixes by designating *fresh* values (that are chosen to be different from all other values) and necessarily equal values. For example, the username chosen for registration is fresh, while the one passed to login must be equal to it. This, again, serves to reduce the number of queries.

⁷As defined, SL^λ does not handle automata with variadic inputs. Gayda [Gay24] shows that it can be extended to efficiently handle these, however.

4.3.2 Benchmarking Method

The case study analyzes the worst-case runtime of the algorithm theoretically, but we also want to evaluate its performance empirically. The evaluation question here is not only if SL^λ performs better than the baseline, i.e., SL^* , but what the impact of its individual optimizations is.

While the Automata Wiki benchmark library [Nei+19] is a suitable choice, the models are sufficiently simple that the state-of-the-art SL^* algorithm [Cas+16] can already solve most of them in sub-second time, i.e., we have “outrun” the benchmark set.

We therefore added a case study based on DTLS-Fuzzer [Fit+22], a learner for DTLS implementations, by extracting all intermediate models produced by the software. These are sufficiently difficult to observe effects from the optimizations.

Finally, to analyze scaling effects more precisely, we added synthesized automata. Since the automaton’s graph structure is invisible to the learner and most structurally synthesized automata are semantically simple, i.e., easy to learn, we instead employed the semantic synthesis approach described in Section 5.2. This allows us to tune various semantic properties of the automaton and introduce sub-automata that can contribute to the complexity of the learning problem.

4.3.3 Evaluation Results

To track the impact of each optimization, the evaluation adds the SL^{CT} algorithm, which is essentially SL^λ without the short suffix optimization, and adds a unrestricted suffix mode that disables the respective optimization, which is otherwise applied to all three algorithms. The short prefix optimization is intrinsic to the algorithm design and cannot be disabled.

During the evaluation, we tracked the algorithm’s number of operations, i.e., the number of queries posed to each oracle. This is a standard technique for evaluating active learning automata which makes the results robust to optimizations, CPU speed, etc. However, we also tracked the wall clock time required for executing the algorithm. We will present only a subset of data obtained that is representative for the key results and refer to Paper IV [Die+24; Die+26a] for more details.

The timing and operation results on a subset of Automata Wiki benchmarks are shown in Tables 4.21 and 4.22, respectively. With respect to timing, no clear picture emerges, since, with the exceptions of the larger queue automata, all algorithms require less than one second to learn the benchmark to completion. On the largest benchmark, SL^λ shows a notable improvement over SL^* , which is in turn faster than SL^{CT} . The time required for testing the models is negligible; the remaining evaluation steps will only focus on learning time.

Automaton				Time (Learn)			Time (Test)		
Name	$ Q $	$ \Gamma $	$ X $	SL^λ	SL^{CT}	SL^*	SL^λ	SL^{CT}	SL^*
ABP Channel	5	8	3	0.04 s	0.04 s	0.05 s	0.29 s	0.29 s	0.30 s
ABP Output	30	50	1	0.15 s	0.70 s	0.14 s	0.18 s	0.15 s	0.26 s
ABP Reciever 3	6	10	3	0.07 s	0.25 s	0.15 s	0.28 s	0.26 s	0.26 s
Login	12	19	4	0.05 s	0.07 s	0.09 s	0.30 s	0.31 s	0.30 s
Repdigit Palindrome	6	15	4	0.05 s	0.05 s	0.07 s	0.40 s	0.40 s	0.41 s
SIP	30	72	2	0.35 s	0.35 s	0.37 s	0.15 s	0.16 s	0.19 s
Queue (size 3)	12	16	4	0.11 s	0.11 s	0.11 s	0.64 s	0.56 s	0.55 s
Queue (size 5)	18	24	6	1.14 s	1.45 s	1.30 s	0.60 s	0.58 s	0.58 s
Queue (size 7)	24	32	8	279.89 s	346.90 s	317.44 s	0.59 s	0.58 s	0.59 s

Table 4.21: Time required by SL^λ , SL^{CT} , and SL^* on the Automata Wiki

When examining the number of operations, the picture is clearer: the number of queries during learning is noticeably less for SL^λ , while the performance of SL^{CT} and SL^* is close to identical. This reduction comes at the cost of requiring more counterexamples (which are generated using queries). In some cases, the reduction in learning queries outweighs the ones required for counterexample generation, but not on all benchmarks.

On the DTLS case study, the three algorithms are differentiated far more clearly, both in time and in number of learning queries. The number of counterexamples required is similar for all three algorithms and is not shown here. [Figure 4.6](#) shows the time required for learning, plotted against the number of transitions in each model. Consistently, SL^λ is slightly faster than SL^{CT} , which is much faster than SL^* . The former two scale better, with SL^* appearing to require time linear to the number of transitions, while the time required for the others remains roughly constant.

Next, we consider the number of operations shown in [Figure 4.7](#). When using the restricted suffix optimization (cf. [Figure 4.7a](#)), SL^λ and SL^{CT} require a similar amount of operations, while SL^* requires more. If the optimization is disabled, however (cf. [Figure 4.7b](#)), the number of operations is massively increased for all three algorithms: with restricted suffixes, SL^* required less than 30 000 operations at worst, without, it needs over 180 000 in one case. In general, SL^λ now outperforms SL^{CT} , which strongly outperforms SL^* . All these observations are consistent with all three optimizations being beneficial, suffix restriction being the most impactful.

Finally, we consider the results on the semantically synthesized automata, shown in [Figure 4.8](#). While the algorithms are not well-differentiated w.r.t. time, the number of operations relative to the number of transitions shows a differentiation both when using restricted (cf. [Figure 4.8a](#))

Automaton Name	Queries (Learn)			Queries (Total)			Counterexamples		
	SL^λ	SL^{CT}	SL^*	SL^λ	SL^{CT}	SL^*	SL^λ	SL^{CT}	SL^*
ABP Channel	11	15	11	28	32	24	2	2	1
ABP Output	208	311	363	4 552	6 151	590	11	11	5
ABP Reciever 3	88	466	489	249	610	614	4	4	4
Login	244	433	436	300	512	509	2	3	3
Repdigit Palindrome	358	476	479	384	504	508	5	5	5
SIP	233	345	487	3 633	2 772	934	15	16	9
Queue (size 3)	24	23	29	202	209	212	5	5	5
Queue (size 5)	55	60	66	434	468	435	7	7	6
Queue (size 7)	96	123	118	839	989	738	8	9	7

Table 4.22: Operations generated by SL^λ , SL^{CT} , and SL^* on the Automata Wiki

and unrestricted (cf. Figure 4.8b) suffixes. Again, removing the optimization results in an increase in the number of queries (here, by a factor of ca. 2). SL^λ slightly outperforms SL^{CT} in the unrestricted case, SL^* requires far more operations in all cases. While consistent with our hypotheses, the lack of differentiation between SL^λ and SL^{CT} may be due to limitations of the synthesized automata, which do not substantially benefit from the short suffix optimization.

We can infer that not only is SL^λ superior to the baseline, SL^* , but can estimate the impact of the three optimizations: suffix restriction appears to be most impactful, followed by short prefixes, followed by short suffixes. Suffix restriction appears to “drown out” the benefits of short suffixes in some cases. While the results on the Automaton Wiki automata were somewhat inconclusive, they are consistent on all larger scenarios considered.

In summary, the benchmark library contributed least to the empirical content due to the simplicity of its examples. Since the baseline was already capable of solving most examples near-instantly, the observations on relative performance were not highly significant. This highlights the relevance of maintaining such libraries at an appropriate “difficulty level” to avoid becoming too simple for current algorithmic advances. The case study and synthetic automata resulted in similar data due to the extensive size of the case study, which allowed the observation of scaling effects. Their consistency, however, still strengthens the conclusion’s empirical content by indicating that the results generalize to many types of models.

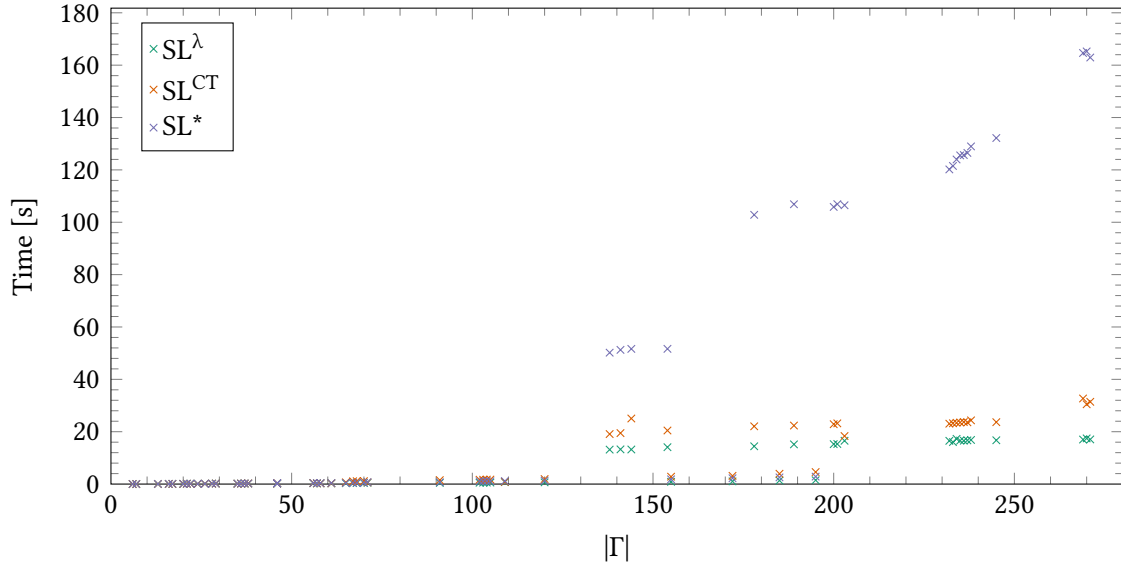


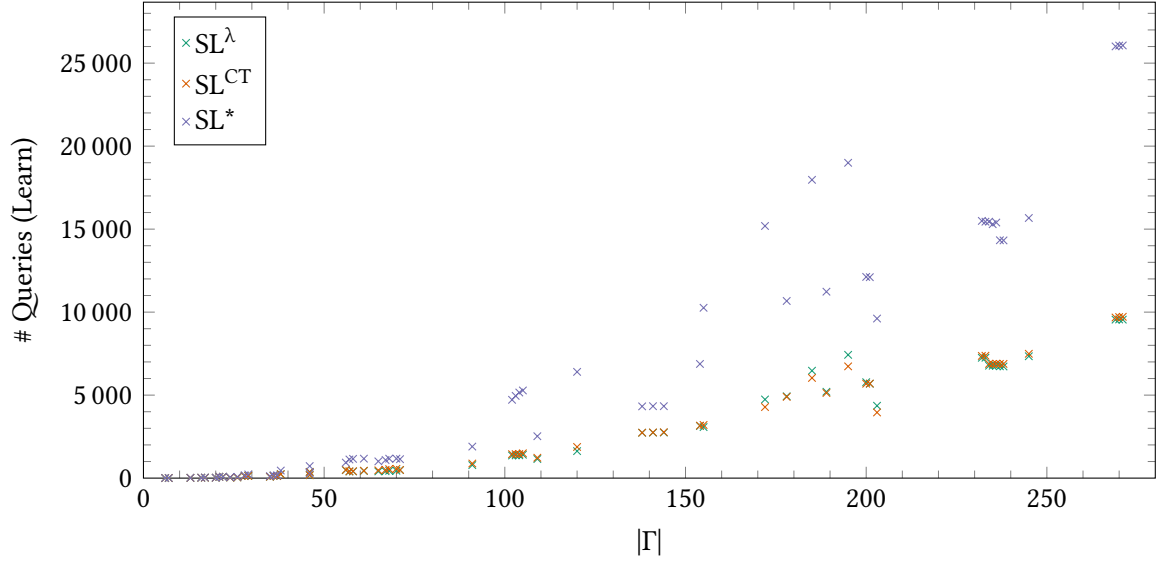
Figure 4.6: Learning time required by SL^λ , SL^{CT} , and SL^* on the DTLS case study

4.4 Case Study III: Intersection Controller Learning

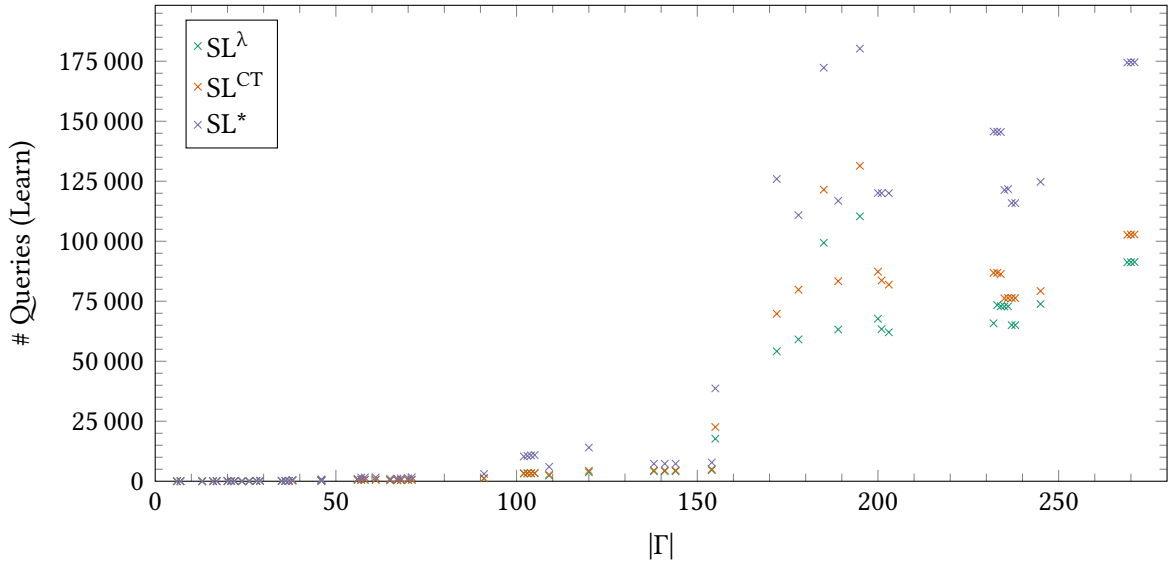
Our third case study studies passive timed automata learning, i.e., derivation of an automaton from traces. This study was motivated by a specific use case: reverse-engineering of the control software for a smart intersection near Vejle, Denmark. However, we only had access to a single, multi-day log of the control software’s inputs and outputs. This can not be handled using existing passive learning algorithms that require either negative examples or a large set of inputs.

We addressed this by implementing an abstraction layer over the trace that facilitated the use of existing learning technologies, both active and passive, resulting in four different learning pipelines. We evaluated these pipelines using the case study at hand as well as a toy example. Like in the previous sections, in [Section 4.4.1](#), we will outline the learning pipelines’ architectures. Next, in [Section 4.4.2](#), we will analyze the choice of benchmarks, and finally, in [Section 4.4.3](#), we will summarize the evaluation results and review the impact of benchmark selection w.r.t. the study’s empirical content.

This section is based on [Paper III \[Die+23a\]](#), but considers its results from a benchmarking perspective, focusing on benchmark selection and its impacts.

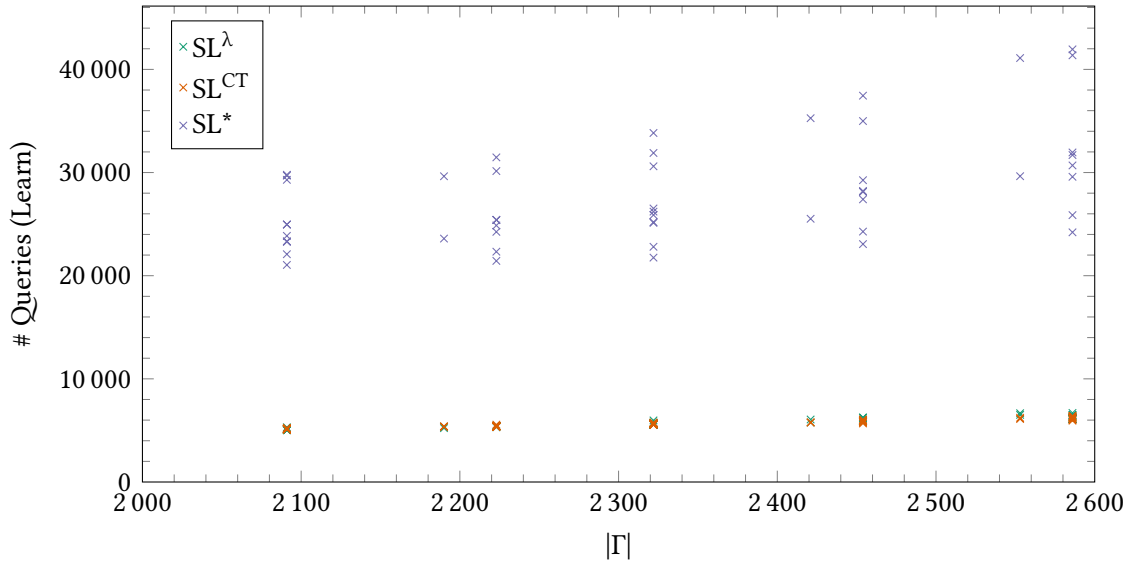


(a) restricted suffixes

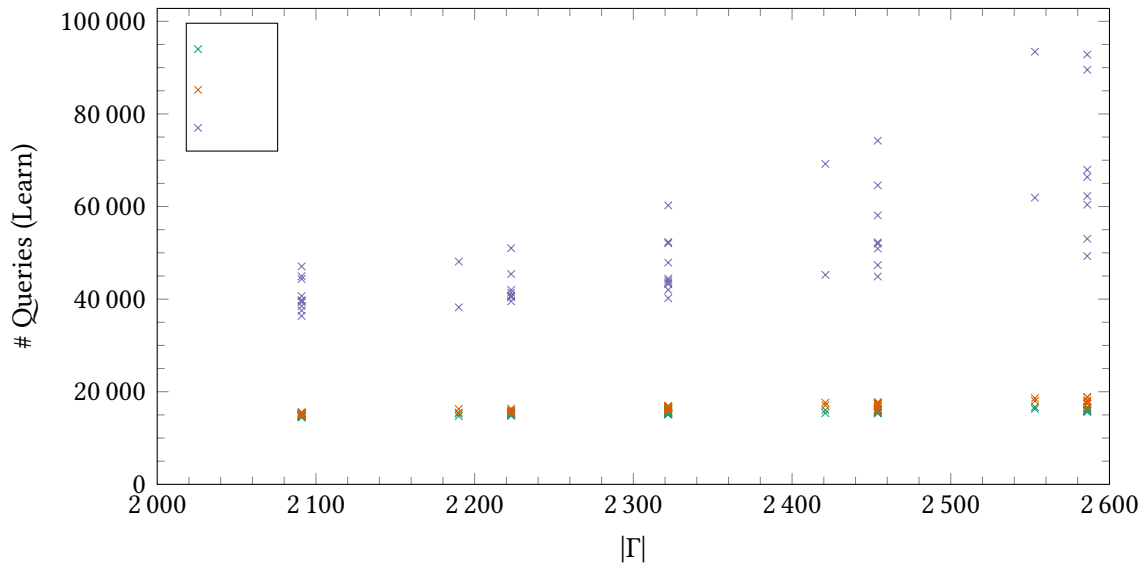


(b) unrestricted suffixes

Figure 4.7: Resets generated by SL^λ , SL^{CT} , and SL^* on the DTLS case study



(a) restricted suffixes



(b) unrestricted suffixes

Figure 4.8: Resets generated by SL^λ , SL^{CT} , and SL^* on semantically synthesized automata

4.4.1 Background and Approach

As opposed to the previous case studies, a register automaton is not a suitable model for representing the controller. There is neither an infinite alphabet, nor are the semantics defined by remembering inputs. Instead, we know that the control software changes the signal configuration based on traffic detected via radar and elapsed time. Such a model can be represented using a Mealy machine with one timer [VBE21], which combines a Mealy machine [Mea55] with an internal state for tracking time. We use a slightly different notation than Vaandrager et al. and omit some technical aspects. The full definition is given in Paper III [Die+23a].

Definition 6 (Mealy Machine with One Timer). Given an *input alphabet* Σ and an *output alphabet* Ω , a *Mealy Machine with One Timer (MM1T)* is a tuple (Q, q_0, t_0, Γ) , comprising

- a set of *locations* Q ,
- an *initial location* $q_0 \in Q$,
- an *initial timer value* $t_0 \in \mathbb{Z}_{\geq 0} \cup \infty$, and
- a set Γ of *transitions* $\langle q, \sigma, \omega, r, q' \rangle$, each defining
 - a *source location* $q \in Q$,
 - an *input symbol* $\sigma \in \Sigma \cup \{\text{timeout}\}$ for a timeout value $\text{timeout} \notin \Sigma$,
 - an *output symbol* $\omega \in \Omega \cup \{-\}$, for an no-output value $- \notin \Omega$,
 - a *reset* $r \in \mathbb{Z}_{\geq 0} \cup \{\infty, \perp\}$, for a no-reset value $\perp$, and
 - a *target location* $q' \in Q$,

which we conventionally render as $\sigma/\omega, r$.

Intuitively, a MM1T contains a timer that is constantly counting down to zero. If it is set to ∞ , it is disabled. The MM1T is given input with a certain delay between symbols. If the next symbol arrives before the timer elapses, the MM1T behaves like a Mealy machine and selects the appropriate transition. If not, it takes the `timeout` transition. Additionally, each transition defines a reset that either sets the timer to the defined value or leaves it as-is on reset $\perp$.

Figure 4.9 shows an example of a MM1T based on to the UPPAAL brick sorter [Ive+00], a toy example used previously to evaluate the verification of LEGO® MINDSTORMS™ programs. This is a system for sorting toy bricks placed on a conveyor belt. A sensor registers red or black bricks via the respective inputs. Three time units later, a robotic arm physically kicks red bricks from the belt into a collection bucket; black bricks pass into a different bucket at the belt’s end. Note that the system is intentionally defective: feeding bricks into the system too fast overwrites the “stored” classification and mishandles the brick on the belt.

Three major approaches for learning the MM1T exist. However, each approach cannot be applied to the scenario at hand. Active learning is impossible, since there exists no membership oracle. For passive learning, approaches like RPNI [OG93] require both positive and negative

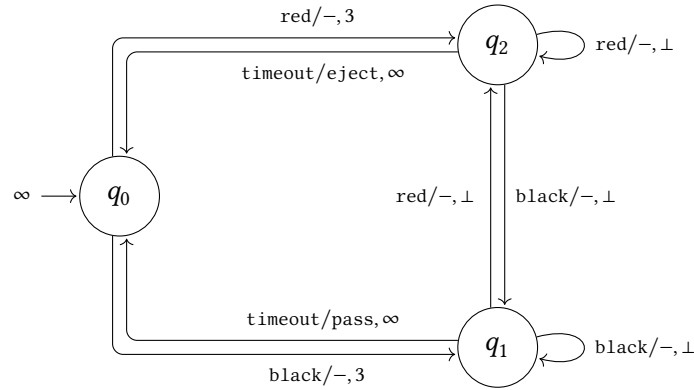


Figure 4.9: A MM1T for the UPPAAL brick sorter example

examples. Since we can not meaningfully create examples of incorrect behavior, these can not be applied, either. Finally, passive approaches like evidence-driven state merging [LPP98] need only positive examples, but require larger input sets to properly differentiate locations.

We enable use of the first and third approach by constructing a symbolic abstraction over the long trace that can be used to answer queries by finding “sufficiently similar” runs in the trace. This can be used to drive active learning. Alternatively, all runs can be enumerated to drive a positive-example passive learner. We will sketch the learning pipelines constructed using this approach and give intuitive explanations for their components, but refer to Paper III [Die+23a] for the implementation details and formalisms.

Since the real controller is adaptive, i.e., continuously updates its internal logic, it is impossible to learn a static automaton model compliant with the trace. Instead, we created a digital twin of the intersection by modeling a controller using UPPAAL⁸ [LLN18a; LLN18b]. Then, as a first step in all pipelines, we reprocessed the real traffic data by using it as an input for the twin controller. The resulting trace combines real traffic data and the digital twin’s signal settings.

The second step of all learning pipelines applies domain knowledge of the intersection: if the signals are in a certain state and no vehicles have recently been detected, it is in a base state; we can split off a run there. Additionally, we expect the intersection to ignore repeating signals, since, e.g., if one *or more* vehicles are present, it has to permit them to pass. This allows us to create the *trace database*, an abstraction over the trace that a) splits data into runs starting in the same internal state and b) can answer queries using sufficiently similar runs.

Instead of querying the database for a specific word with concrete timings, we define a symbolic query language that can be used to retrieve “similar” runs from the database. While a normal oracle would be queried using input symbols with concrete timestamps, the trace database accepts inputs *and outputs* with symbolic timestamps (i.e., variables) instead. In

⁸<https://uppaal.org/>

addition, the asker can specify a constraint over the symbolic timestamp. For example, the query $\langle \text{black}, t_0 \rangle, \langle \text{red}, t_1 \rangle$ with constraint $t_0 < t_1$ could be answered by $\langle \text{black}, 0 \rangle, \langle \text{red}, 2 \rangle$.

Since a MM1T learner will pose concrete, not symbolic queries, we require a *symbolic oracle* that converts concrete membership queries to symbolic ones. The oracle caches prior answers and queries in a tree structure (this is, again, a MM1T) and will only extend this tree when the need arises. If the trace database can not find an empty for a query, the oracle will instead assume that the automaton under learning does not have the corresponding transition to ensure that some answer is given. For example, when posed the concrete query $\langle \text{black}, 0 \rangle, \langle \text{red}, 1 \rangle, \langle \text{timeout} \rangle$, we would discover that the first two symbols can already be answered from result of the symbolic query above. We would therefore extend it with the wildcard $*$ to $\langle \text{black}, t_0 \rangle, \langle \text{red}, t_1 \rangle, \langle *, t_2 \rangle$ with $t_0 < t_1 \wedge t_2 = 3$. The timeout of 3 would have previously been discovered by the learner. The answer would then be $-, -, \text{eject}$.

We can now apply the combination of trace database and symbolic oracle to construct four learning pipelines plus one “baseline” pipeline, an overview of which is shown in Figure 4.10.

- (a) The first pipeline performs active learning by applying the TTT algorithm [IHS14]. TTT’s counterexample analysis results in shorter queries than, e.g., L^* [Ang87], mitigating issues with missing information in the trace database. TTT’s membership queries can directly be answered by the symbolic oracle and we simulate the equivalence oracle by testing random runs.
- (b) The second pipeline extends the first by refining the resulting model using the Paull-Unger minimization algorithm [PU59]⁹. Since our implementation considers missing transitions to be compatible to existing ones for purposes of merging locations, this may slightly alter the automaton’s accepted language.
- (c) The third pipeline applies passive learning instead of active learning. The tree maintained by the symbolic oracle is, by design, a prefix tree acceptor-like structure, which can be extracted from the oracle by brute-force enumeration. Most passive learning algorithms operate by first consolidating their observations into a prefix tree acceptor, which is then generalized by merging locations. We can therefore apply the latter step to the oracle’s tree. In this pipeline, we use the same Paull-Unger minimizer [PU59] as in the last approach.
- (d) Finally, the fourth pipeline only exchanges the learning algorithm, using an adapted k -Tails algorithm [BF72] instead of Paull-Unger. Since k -Tails infers a nondeterministic automaton, we determinize the learned automaton afterwards.
- (e) As a baseline for the evaluation, we also implemented a simplified version of the TkT algorithm [Pas+22] that does not consider operation durations and therefore can not

⁹While optimized versions such as Hopcroft’s [Hop71] exist, this is sufficiently fast for our experiments.

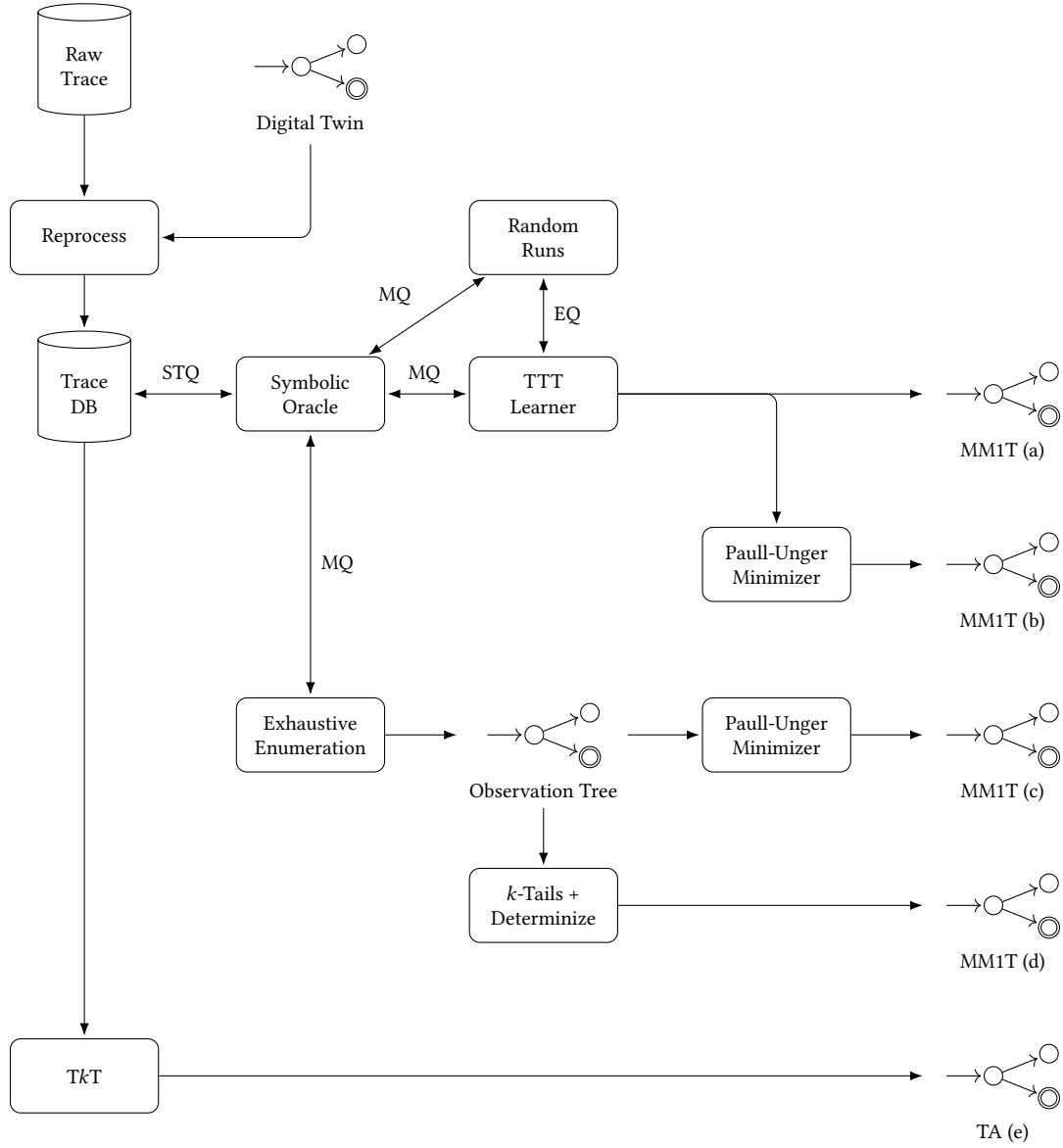


Figure 4.10: Learning setups for reverse-engineering the intersection controller. The five setups' results are designated (a)–(e)

infer clock resets. This algorithm creates a timed automaton without resets and can operate directly on the runs split by the symbolic oracle.

4.4.2 Benchmarking Method

The case study’s algorithmic contribution is a novel method to handle unstructured data. The evaluation therefore focuses on showing the method’s feasibility in the specific case and is not concerned with showing competitiveness.

Since the research project was motivated by a specific case study, its use in the evaluation is a natural choice. Since we had created a digital twin to reprocess the trace, we also had access to a ground truth model. This would not be the case for the original data. We augmented the case study with a relatively complex toy example for UPPAAL, a brick sorter [Ive+00], for which we also had access to a ground truth automaton we could use for benchmarking.

To the best of our knowledge, no benchmark libraries for MM1Ts existed at that point. While libraries of timed traces, e.g., the LogHub [Zhu+23] Hadoop server log collection [Lin+16] exist, these do not have a ground truth automaton, contain extremely large logs, and would additionally have required a domain-specific reimplementaion of the segmentation logic.

We did not attempt to synthesize any automata for this project. Because the evaluation is not focused on scaling behavior, the need for such benchmarks is lessened. Also, we are not aware of any suitable synthesis techniques for MM1Ts aside from random generation.

4.4.3 Evaluation Results

To compare the five pipelines, we executed them on traces generated on the brick sorter toy example and the (reprocessed) trace obtained from the case study. To observe some scaling effects, we also generated reduced versions of the latter that only considered a subset of the lanes and signals. Approach (e), the TkT learner, can not be easily applied to the reduced versions.

For all traces, we considered the accuracy of the final model w.r.t. the symbolic tree (i.e., how many runs *in the tree* are accepted by the final model) as well as the human-readability of the model via manual inspection. Since approach (e) is a state-merging algorithm that generalizes from the inputs, it cannot misclassify them and will always achieve a perfect accuracy.

We show the quantitative results in Table 4.23. It is apparent that with increasing size and complexity of the system under test, the accuracy of all approaches decays, while the model size increases. For the brick sorter, approaches (a)–(d) perfectly recovered the MM1T model, including the classification bug present in it. On both models, pipeline (e) generates far larger

Data Set			Accuracy					Q				
Name	$ \Sigma $	LST	(a)	(b)	(c)	(d)	(e)	(a)	(b)	(c)	(d)	(e)
Brick Sorter	2	139	100 %	100 %	100 %	100 %	100 %	3	3	3	3	7
Intersection I	3	638	99 %	99 %	86 %	100 %	n/a	2	2	19	4	n/a
Intersection II	4	581	97 %	98 %	99 %	98 %	n/a	5	2	2	6	n/a
Intersection III	5	970	86 %	83 %	81 %	97 %	n/a	9	7	29	11	n/a
Intersection IV	8	1 351	93 %	93 %	77 %	97 %	n/a	7	3	37	10	n/a
Intersection All	11	1 839	81 %	81 %	77 %	94 %	100 %	13	13	50	23	749

LST: leaves in the symbolic tree n/a: TkT cannot be used in this scenario

Table 4.23: Accuracy and result automaton size of all intersection learning pipelines

models than pipelines (a)–(d), which also were far less human-readable on manual inspection due to size and the lack of reset information.

Among the remaining four approaches, pipeline (c), the Paull-Unger minimized symbolic tree, performs noticeably worse than the other three on all but the Intersection II data set (which was learned very well by all approaches), yielding larger but inaccurate models. The performance of pipelines (a) and (b) is similar, with (b) yielding slightly smaller models without accuracy loss. Between the pipelines (b) and (d), there is a visible trade-off between smaller, less accurate models produced by active learning and larger, more accurate models produced by k -Tails.

We can conclude that for the given examples, pipeline (d) (and, to a lesser extent, (b)) are feasible approaches to learning from a single, unstructured trace. The scaling behavior matches expectations, giving rise to some confidence in the results. However, it is not clear to what extent the result holds for other scenarios.

In summary, the benchmarks are sufficient to show the feasibility of the approach and its susceptibility to scaling. The addition of the brick sorter toy examples helps show that the result is not restricted to the intersection example. Still, the benchmark combination severely limits the empirical content, since strong predictions about other scenarios cannot be made from the limited data. Such conclusions would require future work on more benchmarks.

4.5 Case Study IV: Heat Pump Learning

The fourth case study also concerns use-case driven passive learning from real-world data. In this study, we reverse-engineered a model of a domestic heat pump from a single log spanning multiple days. As opposed to the intersection controller, these logs contain continuous-valued data, e.g., temperature readings. Such a model could then be used to predict, e.g., good

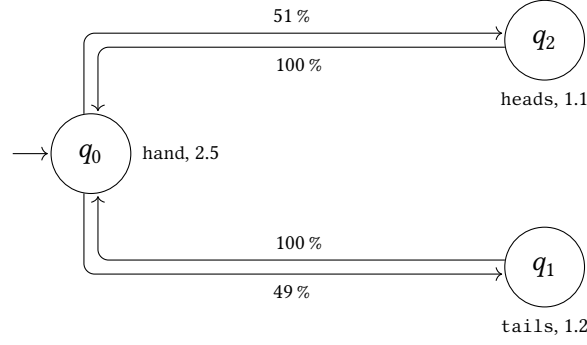


Figure 4.11: A CTMC for a biased coin flip

opportunities for running a defrosting cycle. As opposed to the previous study, however, we did not use domain-specific information in developing the learning approach.

We developed a learning pipeline that is parameterized using a metaheuristic, eliding the need for domain-specific parameter selection. Since no suitable evaluation metric for our chosen model exists, we additionally provide a novel similarity metric for traces and automata. We evaluated the evaluation metric using synthetic automata and the learning pipeline on both the motivating case study and a toy example. As in previous case studies, we first outline our learning pipeline and the novel metric in [Section 4.5.1](#). We then discuss the choice of benchmarks in [Section 4.5.2](#) and will summarize the evaluation results and review the impact of our chosen benchmark types in [Section 4.5.3](#).

This section is based on [Paper VI \[Die+26b\]](#), but considers its results from a benchmarking perspective, focusing on benchmark selection and its impacts.

4.5.1 Background and Approach

In this case study, we learned from a mixture of series of both continuous- and discrete-valued time series. We identify each series by a *variable* (e.g., temperature data). However, our ultimate model used discrete data only to allow for the less complex automata. The full model therefore consists of a set of discretization rules to convert continuous values and an discrete model for these. While this combination is capable of recognizing continuous-valued series, it is incapable of generating them.

In summary, we require an automaton modeling timing semantics over a finite domain. As opposed to the [previous case study](#), the data also contains the results of (nondeterministic) human interactions; additionally, the discretization step will likely introduce ambiguities. As a result, we have to infer a probabilistic automaton model instead of a deterministic one. We chose (deterministic) continuous-valued Markov chains [[Mao+16](#)] as a formalism. As opposed to MM1Ts, these do not have a global timer, but a per-location one. They do not accept inputs,

but generate a stream of outputs with delays defined by the timers. We simplify some aspects of this definition and refer to [Paper VI \[Die+26b\]](#) for the full version.

Definition 7 (Continuous-Timed Markov Chain). Given an *output alphabet* Ω , a *continuous-timed Markov chain (CTMC)* is a tuple (Q, q_0, λ, p, t) , comprising

- a set of *locations* Q ,
- an *initial location* $q_0 \in Q$,
- an *output function* $\lambda : Q \rightarrow \Omega$,
- a *transition probability function* $p : (Q \times Q) \rightarrow [0, 1]$, and
- a *average exit time function* $t : Q \rightarrow \mathbb{R}_{\geq 0}$.

We require that for all locations $q \in Q$,

- the sum $\sum_{q' \in Q} p(q, q') = 1$ and
- for all outputs $\omega \in \Omega$, at most one location q' with output ω has transition probability $p(q, q') > 0$, i.e., successors are deterministic w.r.t. their outputs. We call CTMCs meeting this criterion *deterministic*.

Intuitively, a CTMC produces an infinite stream of timed outputs. In each location, the output symbol is emitted. The CTMC then waits a period of time determined by the location’s average exit time, nondeterministically selects a successor according to the probability function, and repeats this process. Additionally, a sequence of timed outputs can be unambiguously fitted to a sequence of CTMC locations, since each location can transition to at most one successor for any given subsequent output.

[Figure 4.11](#) shows an example of a CTMC modeling a slightly biased coin flip. On average, the coin is retained in the thrower’s hand for 2.5 time units. In 51 % of cases, the coin shows heads and is picked up in 1.1 time units on average, in 49 %, it shows tails and is picked up in 1.2 time units. Note that a sequence such as `hand, heads, tails` is not admitted by the model.

Before we can attempt infer a CTMC, we need to transform the continuous-valued inputs into a finite output language using a *discretizer*. We apply the established PELT algorithm [\[KFE12\]](#) to independently segment each continuous-valued variable’s series into segments with “similar” behavior (e.g., “value oscillating between 3 and 5”). Each behavioral class then corresponds to a output symbol. PELT requires several parameters to be set depending on the data analyzed.

Next, we add an additional simplification step to remove redundant (e.g., the readings from multiple temperature sensors) or irrelevant (e.g., network diagnostics) variables from consideration and merge the remaining variables’ discretized data into combined traces. If the previous step has identified likely reset points, we can use these to segment the merged trace, alternatively, we can segment by time or generate a single, long trace. The choice of variables is passed as a parameter to this component.

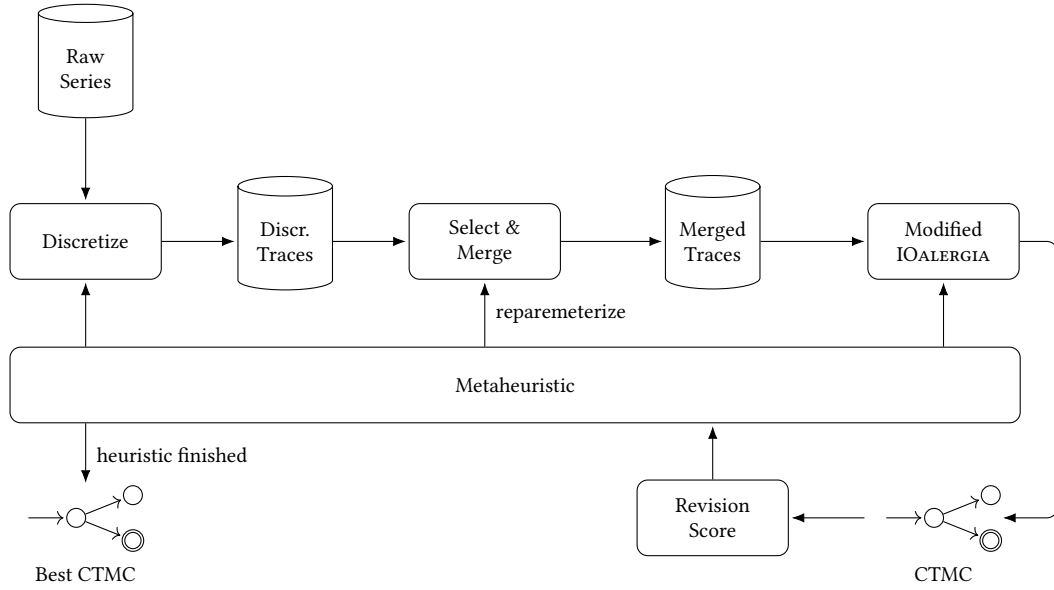


Figure 4.12: Learning pipeline for reverse-engineering CTMCs from arbitrary systems

Finally, we can learn an automaton from the discretized and merged data. The timed IOALERGIA [Mao+16] is an established approach for learning CTMCs from traces. Algorithms in the ALERGIA family are built on the notion of similarity: starting from a prefix tree acceptor, two locations are merged if a) their outputs are identical, b) their timing and transition probability behavior are sufficiently similar and c) for all outputs, the respective successor locations are eligible for merging, too.

Intuitively, sufficient similarity is defined based on the number of observations: for example, probabilities of 20 % and 25 % are similar when based on four and five observations, respectively, but likely dissimilar when based on a million observations each. This intuition is formalized by applying the Hoeffding [Hoe63] and F-tests [Cox64] to transition probabilities and timings, respectively. Each test yields a similarity score, which can then be compared against a threshold chosen by the user. We additionally define three extensions for IOALERGIA:

Merged data IOALERGIA, as defined by Mao et al., performs all stochastic tests on the observations in the prefix tree acceptor and does not consider “merged-in” data. This is necessary for proofs of convergence behavior to hold. When using very sparse data, i.e., in this case study, this effectively disables all tests. Based on a suggestion by Mao et al. [Mao+16], we allow using the merged data instead.

Test depth Demessance et al. [Dem+21] define the *relaxed* ALERGIA algorithm that does not recursively perform the stochastic tests. We extend the approach to define a test depth that, similar to k -tails [BF72], only performs tests to a defined recursion depth.

Threshold decay Generalizing from this idea, we also allow for threshold decay. By selecting a decay factor for a type of test, the threshold for that test is exponentially decayed in each recursion step.

Finally, we need to evaluate the resulting model’s quality. Unfortunately, the literature only describes algorithms that compare two CTMCs (requiring a “ground truth” for the input trace) or cannot be applied in a continuous-timed scenario. We define the *revision score* metric to fill this gap. It compares a set of traces to a given CTMC and yields score in $[0, 1]$. Intuitively, it “replays” the trace against a CTMC learned using IOALERGIA and computes the Hoeffding and F-test scores between the observations in the trace and those resulting from IOALERGIA’s merging from the prefix tree acceptor.¹⁰ We can then average these scores to a single value.

We define three versions of this score. The *single initialized* mode replays each validation trace individually, starting in the initial location. All scores are then averaged. The *single best* mode tries every location as a starting location instead, selects the best score per-trace and averages these. Finally, the *global* mode replays all validation traces from the initial state and then performs the stochastic tests once.

All of the learning steps require parameters that are highly domain-dependent. Instead of setting them manually (and requiring domain knowledge to do so), we integrate the learning process into a metaheuristic optimization process. The resulting pipeline is shown in Figure 4.12. We use two loops of particle swarm optimization [KE95] as an optimization algorithm, with the outer loop parameterizing the discretizer and the inner loop the two other steps. The revision score is used as a cost function.

4.5.2 Benchmarking Method

The main algorithmic contribution of the case study are two methods: the learning pipeline and the revision score. Similar to the [previous case study](#), the pipeline enables learning in cases that could previously not be handled and is consequently only evaluated for feasibility and not competitiveness. It was therefore evaluated using the motivating case study, however, in this case, no ground truth model was available. We therefore have to use the limited data set for both learning and validation. We also added a toy example, this time modeling a buck-boost converter without having access to a suitable ground-truth model.

However, the evaluation of the revision score requires showing a similarity to established CTMC comparison metrics for automaton-automaton comparisons. We therefore require ground truth models and sufficient samples to show a correlation, i.e., the motivating case study and the toy example are insufficient for this task. To obtain two ground-truth models and a larger data set, we used structural synthesis to generate CTMCs for this evaluation step.

¹⁰This requires storing the relevant information in addition to the probabilities. We always use the merged observations in this step.

The generation uses a naïve structural synthesis approach, since more complex generation approaches do not exist to the best of our knowledge. Since the learning pipeline requires *continuous*-valued data as input, however, the generated CTMCs can not be used in the pipeline’s evaluation.

Similar to the intersection case study, we did not use benchmark libraries like LogHub [Zhu+23]. Without a ground truth, the evaluation results are difficult to interpret and execution of the pipeline in large datasets is highly time-consuming. We instead focused on performing a rigorous evaluation on the case study and toy example described above.

4.5.3 Evaluation Results

To establish the use of revision scores in the learning pipeline, we first compared this metric to the one proposed by Mao et al. for timed IOALERGIA [Mao+16]. They first define a property using subCSL, a linear fragment of CSL [Bai+03]. Starting from two CTMCs, they generate a large set of random traces for each and measure the share of traces satisfying the property. If the shares are similar, the CTMCs are. We extend this approach to the average satisfaction difference for a large number of random properties, which we call *subCSL score*. Note that the number of traces required exceeds the data available in the case study by far.

For pairs of random automata and traces generated from them, we compared the subCSL score of the trace pairs and the revision score of between one CTMC’s trace and the other automaton. We expect these values to correlate to show the feasibility of the revision score metric. The results of the experiment are shown in Figure 4.13 for two output alphabet sizes. The results from both single modes are identical and hint at a linear correlation between revision score and subCSL score. The global mode appears to be less useful, with values close to zero in many cases. We conclude that single-mode revision scores seem suitable as a similarity metric in our use case.

We then evaluated the learning pipeline using both the heat pump case study and the buck-boost toy example. We tested all modes for the revision score metric and use the score of the metaheuristic’s final output as a measure of success. The results are shown in Table 4.24. In general, the single best mode shows the best test performance on the heat pump data set and a good correspondence between training and test scores on all data, outperforming the single initialized mode. The global score yields perfect scores on the heat pump training set which are not achieved in the test set, raising doubts about its suitability on realistic data. Scores on the test data set were highly satisfactory for the heat pump, falling between 80 and 85 % for the single best mode. On the buck-boost toy example, the single mode scores were exactly two thirds for both single modes, which is also acceptable.

Our observations w.r.t. the choice of benchmarks are similar to those in previous case studies: to show a correlation between two measures, synthesis is a suitable approach, while case studies and toy examples can serve to establish an algorithmic approach’s feasibility. However,

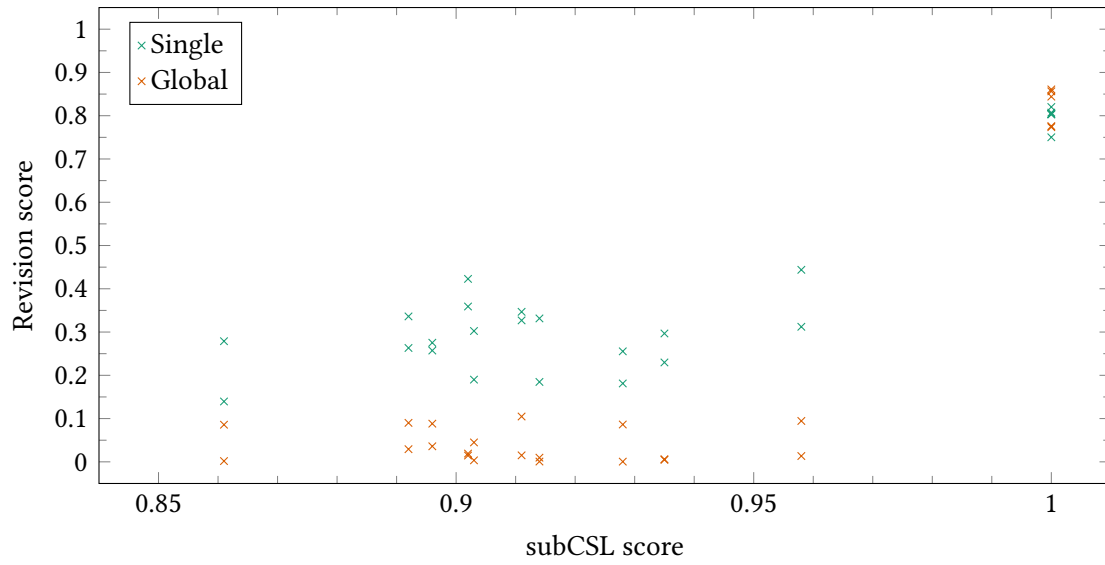
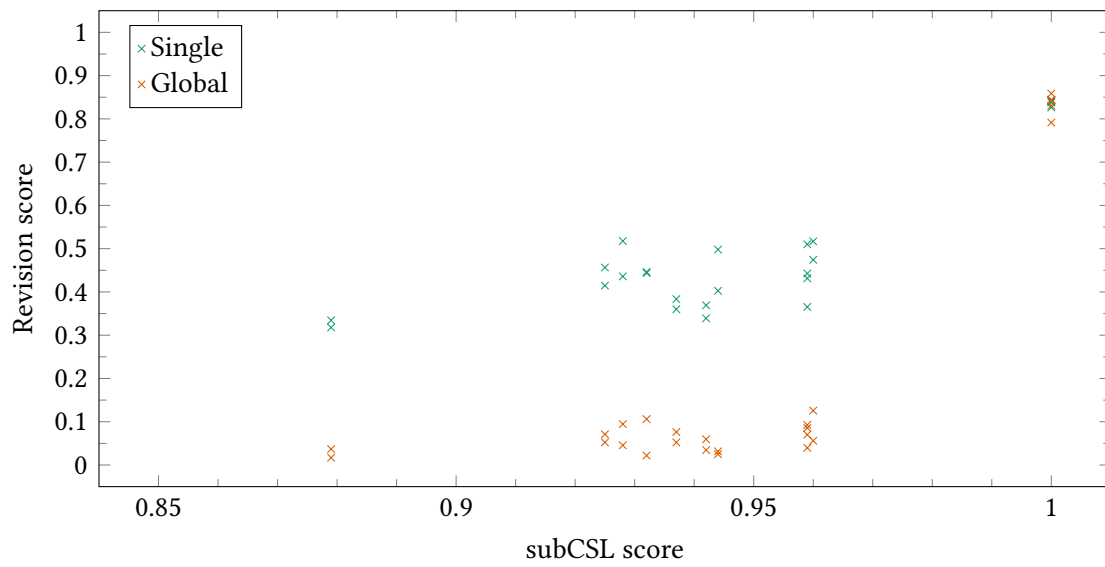
(a) $|\Omega| = 3$ (b) $|\Omega| = 6$

Figure 4.13: subCSL vs. revision scores on synthetic automata

Data Set	Variables	Events	Traces		Rev. Score		RS Mode
			Train	Test	Train	Test	
Buck-Boost Converter	3	186 727	7	3	57 %	67 %	Single Best
Buck-Boost Converter	3	186 727	7	3	57 %	67 %	Single Initialized
Buck-Boost Converter	3	186 727	7	3	100 %	100 %	Global
Buck-Boost Converter	3*	186 727	7	3	57 %	48 %	Single Best
Heat Pump	10	332 412	4	1	85 %	80 %	Single Best
Heat Pump	18	332 412	22	11	79 %	83 %	Single Best
Heat Pump	18	332 412	22	11	72 %	60 %	Single Initialized
Heat Pump	18	332 412	22	11	100 %	77 %	Global

*: no variable removal

Table 4.24: Best revision scores on non-synthetic models

both results have only limited empirical content due to missing benchmarks of the other type: the revision score result is not established on any real-world data, while the pipeline is only evaluated on two use cases, showing neither generalization nor scaling effects. Here, too, more benchmarks would be needed to address this gap.

4.6 Discussion

Existing Works In research on FSAs, a clear evolution of benchmarking can be seen. Initially, only theoretical works were created that set the foundation of the field. Next, case studies started being used to perform empirical evaluations and compare the individual algorithms. This progression is apparent for both active and passive automata learning. Case studies supplant toy examples around the year 2000, which may be indicative of increased availability of both processing power and systems usable as case studies. Around the same time, synthetic benchmark start being used, perhaps also due to the increase in available processing power. Benchmark libraries merit a closer examination: while they start being created and used in the 1990s, a lifetime of benchmark libraries becomes apparent, with each library eventually disappearing from the literature. However, their use appears to be the norm in recent work. Combined approaches are also being used increasingly often.

EFSA research seems to have skipped the first two stages of this evolution, seeing almost no use of toy examples and very few theoretical works. This may be attributable to the field being established after 2000, when toy examples appear to have been obsoleted in FSA research. However, we can not observe substantial use of benchmark libraries, with the Automata Wiki [Nei+19] being the only library containing EFSAAs at all (namely, succinct RAs). This may

be caused by the plethora of EFSA models that are not convertible between each other (see [Chapter 3](#) for an example). Synthesis is also used sparingly, mostly to augment case studies and toy examples. A reasons for this might be the lack of synthesis algorithms for EFSA beyond naïve random generation. In summary, EFSA research appears to be somewhat “stuck” in a case study phase, inherently limiting the empirical content of evaluations performed there.

In both lines of work, comparative evaluations are often somewhat limited. Benchmarking is usually performed against a state-of-the-art algorithm, however, the benchmark set used may be fundamentally different than the one used for that algorithm’s claims. In the benchmarking of product-L*, it became apparent that the authors’ claimed improvements did only consistently manifest on toy examples, but not a benchmark library, showing the potential threats to validity of that approach. This can also be seen in the comparison study by Aichernig et al. [[ATW24](#)], where differences between learning algorithms were shown to be highly dependent on the learning setup and not only – as indicated by some previous publications – the algorithms themselves.

Threats to Validity The validity of this analysis is potentially limited by two factors: the informal nature of the literature survey and the approach to coding, i.e., assignment of benchmark categories to the surveyed publications. The survey was created using an informal approach, starting with publications known to us and focusing on the field of automata learning. Ideally, the structured literature review method (cf. Carrera-Rivera et al.’s method overview [[Car+22](#)]) would be used to ensure a better coverage. Since backwards search ceased to yield any unknown publications, we are fairly confident that our coverage of automata learning-related works is good, however, this is likely not to be the case for automata-related work in other fields. This may limit the external validity of our conclusions w.r.t. other fields.

The second issue stems from the fact that benchmarking categories were defined from subjective experience and then assigned to publications by us. This approach to coding may introduce researcher bias, which could be reduced by, e.g., employing independent coders. For an in-depth discussion of coding methods, we refer to Seaman’s method overview [[Sea99](#)] and Miles et al.’ textbook [[MHS20](#)] on the subject. However, in the vast majority of cases, we found the assignment of categories to be extremely clear-cut. Still, reliance on this subjective observation may pose threats to the internal validity of the results.

Case Studies The case studies we conducted support both our observations and concerns and indicate a possible way forward. We can also make observations about benchmarks’ impact on empirical content. The first study (backwards propagation) used both a benchmark library and synthesis, with the combination being essential to the results’ empirical content. While the benchmark library was used for its realistic content, synthesis showed scaling

effects we could not reliably observe on the library. In combination, they support the claim that the approach is efficient on realistic data *and* scales well to large models.

In the second study (SL^λ), our use of a benchmark library was less helpful, since all algorithms we considered were able to learn the automata with ease. This might indicate a reason for the library lifetimes we observed on FSAs: eventually, a library's contents may become too trivial due to improvements in algorithms, making the library obsolete. In the SL^λ case study, we addressed this issue by including an additional case study benchmark that was complex enough to yield significant results on realistic data. Synthesis again served to complement these results with observations on scaling effects. The impact on empirical content is similar to the first case study: in combination, they show that (and why) the algorithm outperforms the state of the art on realistic *and* large automata.

The third and fourth case studies (intersection and heat pump) show the limits imposed by a lack of benchmarks. In both cases, we only had access to a single case study and one toy example. As a result, the study mostly shows the feasibility of the proposed approach and cannot make general claims of performance or quantify scaling behavior on large systems. In the heat pump case study, the evaluation metric was instead evaluated on synthetic automata only, leaving questions about its performance on more realistic models open.

Conclusion We can make some conclusions on the impact of benchmarks. First, case studies and toy examples are insufficient to support claims with high empirical content, since too little data is created. Toy examples are subject to unintended biases by the researchers (by favoring cases in which their approach performs best) and the limits in complexity resulting from their manual creation. Case studies, on the other hand, can not support claims of generalization well. For EFSA research, this means that many evaluations also can only make conditional predictions. Second, synthetic and realistic benchmarks are complementary, with the former excelling at supporting claims of scaling and the latter supporting claims of realism. In isolation, each claim's empirical content is low: either the results are predicated on the synthesis' realism or on the complexity of benchmarks present in the library. When combined, the empirical content effectively becomes the *product* of its parts' content. Therefore, the combination of benchmark libraries and synthetic benchmarks can be defined as the "gold standard" of benchmarking.

In EFSA research, we only see limited adoption of this combination in the past. Addressing these issues requires two lines of work. First, we have observed that benchmark libraries are subject to decay, likely due to advances in algorithms. To keep benchmark libraries relevant, they need to be "living" entities, with new benchmarks representing the current complexity "frontier" being added to the library. This requires adoption of standardized formats and collection of benchmarks in suitable, easily-used libraries. Second, synthesis algorithms for EFSA are still in their infancy. Here, further algorithmic development is needed. In Case Studies I and II, we have already made progress along this line of work.

Chapter 5

Benchmark Synthesis

As established in the [previous chapter](#), synthesis of EFSAs is still a mostly open question and benchmarking techniques are rarely used in the evaluation of algorithms operating on such automata. Structural synthesis approaches are limited to fully random generation, while semantic algorithms do, to our knowledge, not exist. We close this gap by defining both a structural and a semantic synthesis algorithm for one type of EFSA, namely register automata.

In this chapter, we present two algorithms for synthesizing register automata. In [Section 5.1](#), we present the structural algorithm developed for the benchmarking of Case Study I (cf. [Section 4.2](#)) and in [Section 5.2](#), the semantic algorithm developed for Case Study II (cf. [Section 4.3](#)), plus its accompanying RAGe framework. We will describe each algorithm in detail and consider if and how it could be applied to other types of EFSA.

5.1 Structural Synthesis

First, we will present a structural synthesis algorithm for RAs. This algorithm was developed to support the evaluation of the history independence algorithm shown in [Section 4.2](#), which operates on the automaton’s graph structure. Since the goal of the evaluation was to measure the impact of automaton features without semantic impact, the synthesis was intentionally developed to generate semantically “inefficient” register automata, i.e., RAs that have superfluous locations, overwrite registers without ever reading them, etc.

In the synthesis of RA guards, a central challenge is to avoid the generation of unsatisfiable guards. In [Section 5.1.1](#), we will discuss an approach to representing RA satisfiable guards. Next, in [Section 5.1.2](#), we will apply this representation to construct a highly-configurable synthesis algorithm for RAs. In [Section 5.1.3](#), we will draw a short conclusion and discuss if the approach is usable for other types of EFSA.

This section is based on [Paper II \[DH22a\]](#), however, the design of the synthesis algorithm was not described in that publication. We have added some clarifications to the algorithm over the version used.

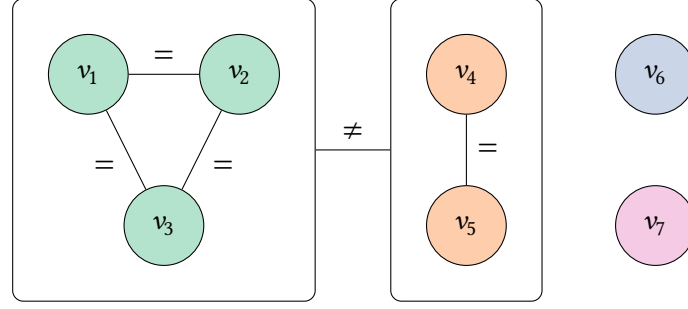


Figure 5.1: Graph representation of the guard $(v_1 = v_2) \wedge (v_2 = v_3) \wedge (v_3 \neq v_4) \wedge (v_4 = v_5)$

5.1.1 Guard Representation

When constructing a register automaton, its transitions need to be equipped with a guard, i.e., we need a process to generate a guard statement (cf. [Definition 3](#)) over a defined set of variables. This process must never yield a guard that is unsatisfiable (i.e., implies $v \neq v$ for a variable) and should not accidentally generate semantically equivalent guards (e.g., $v = w \wedge w = x$ is equivalent to $v = w \wedge v = x$). This excludes the naïve approach to constructing a guard, i.e., randomly select registers and parameters and combine them with equalities and inequalities. Instead, we define a guard representation that is both canonical and satisfiable and can be algorithmically synthesized.

We begin by defining a representation for equalities in a guard g . Assume a set of variables V . Then, the (undirected) *equality graph* $G_{=} = (V, E_{=})$ is defined by $\{v_1, v_2\} \in E_{=} \iff g \vdash (v_1 = v_2)$. This can be constructed by computing the (unique) transitive closure over a guard's equality statements. This graph now defines a partition over V into cliques; in each clique, all variables must be equal. We call the set of partitions $\mathcal{P}_{=} \subset V^2$.

Next, we can observe that inequalities are transitive via equality. Instead of considering inequalities between variable, we can consider inequalities between the partitions induced by $G_{=}$. We can define the (undirected) *inequality graph* $G_{\neq} = (\mathcal{P}_{=}, E_{\neq})$ by $\{P_1, P_2\} \in E_{\neq} \iff g \vdash v_1 \neq v_2 \forall v_1 \in P_1, v_2 \in P_2$. This can be created by transitive closure over the inequalities. Note that if and only if a loop $\{P, P\} \in E_{\neq}$, the guard is unsatisfiable.

In summary, this graph representation is canonical, i.e., equivalent satisfiable guards have the same representation and it is trivial to recognize unsatisfiable guards. If we forbid self-loops on $G_{\neq}$, only satisfiable guards can be described. An example of a graph representation is shown in [Figure 5.1](#) for the formula $(v_1 = v_2) \wedge (v_2 = v_3) \wedge (v_3 \neq v_4) \wedge (v_4 = v_5)$ over the variables $v_1, \dots, v_7$. Note that introducing, e.g., $(v_1 = v_3)$ into the formula will not change the graph representation.

Input: data universe (Λ, D, α)

Input: number of locations $|Q|$, registers $|X|$, accepting locations $|Q^+| \leq |Q|$, and transitions $|\Gamma| \leq |Q|(|Q| - 1)$

```

1 begin
2   generate  $|Q|$  states, of which  $|Q^+|$  (uniformly chosen) are accepting
3   generate  $|X|$  registers
4    $T \leftarrow$  a minimum spanning tree over  $Q$ , rooted in  $q_0$ 
5    $A \leftarrow |\Gamma| - |T|$  random pairs of locations not in the spanning tree
6   convert all edges in  $T \cup A$  into transitions using Algorithm 5.2
7   return the generated automaton

```

Algorithm 5.1: Structural synthesis algorithm for register automata

Input: data universe (Λ, D, α)

Input: a cluster size distribution $\mathfrak{D}$

Input: number of equality partitions $n_=$, inequalities $n_{\neq}$, and assignments $n_{:=}$

```

1 begin
2    $\lambda \leftarrow$  a random label from  $\Lambda$ 
3    $\mathcal{P}_= \leftarrow n_=$  partitions of  $P^\lambda \cup X$  of size  $\mathfrak{D}$ 
4   assign all remaining elements of  $P^\lambda \cup X$  as single-size partitions to  $\mathcal{P}_=$ 
5    $E_{\neq} \leftarrow n_{\neq}$  non-loop edges over  $\mathcal{P}_=$ 
6    $g \leftarrow (\bigwedge_{P \in \mathcal{P}_=, v \in P, v' \in P} (v = v')) \wedge (\bigwedge_{\{P, P'\} \in E_{\neq}, v \in P, v' \in P'} (v \neq v'))$ 
7    $A \leftarrow$  a random subset of  $X$ 
8    $S \leftarrow n_{:=}$  random elements of  $X \cup P^\lambda$ 
9    $a \leftarrow [x := x \mid x \in X \setminus A]; [x := s \mid (x, s) \in A \text{ zip } S]$ 
10  return the transition  $\langle q, \lambda, g, a, q' \rangle$ 

```

Algorithm 5.2: Structural synthesis algorithm for register automaton transitions

5.1.2 Synthesis Algorithm

We can now define a synthesis algorithm for register automata over a given data universe. The algorithm operates in two phases: first, the automaton graph structure is generated. To allow high flexibility, most sizes can be defined by the user to define certain aspects of complexity. The algorithm is shown in [Algorithm 5.1](#). We begin by constructing locations and registers to the desired specification. The graph structure is generated by first creating a spanning tree starting at the initial location to ensure that all locations are theoretically reachable, then “filling in” the remaining edges at random.

In the second phase, we convert the edges created in the previous step into transitions using [Algorithm 5.2](#). We begin by choosing a random label λ (either uniformly, or the user could provide a biased distribution). Now, we can create a guard in graph representation over $P^\lambda \cup X$.

We begin by creating $n_{=}$ equality graph clusters. For each cluster, we sample a user-provided distribution to obtain its size and pick the corresponding amount of variables at random. All non-selected variables are assigned to clusters of size 1. Next, we create $n_{\neq}$ random edges between clusters to form the inequality graph. We can then transform the graph into a guard. In the next step, we create the assignment by selecting $n_{=}$ targets from the register set at random and assigning each a random source from $P^{\lambda} \cup X$. All non-selected registers keep their value. Finally, we assemble the transition and add it to the register automaton.

Note that the numeric parameters of the second phase (e.g., $n_{=}$) can be passed to the first phase as stochastic distributions, i.e., the number of equality partitions can be varied for each edge that is translated to a transition. As a result, the generation is highly configurable.

5.1.3 Conclusion

We have defined a bespoke structural synthesis algorithm for register automata. Its application in Case Study I showed the benefits of its high degree of configurability: evaluations with increasing register and location sizes could be defined and the use of stochastic distributions as parameters allowed variation within the respective automata. The synthesis itself is sufficiently fast to cause no noticeable overhead relative to the algorithm itself.

The structure of the algorithm's first phase is applicable to synthesis of all EFSA models (and FSA models, too). However, the graph-based representation that forms the core of the second phase is highly specific to synthesis of propositional logic expressions and has little relevance to the synthesis of timed automata, for example. It should be possible to apply the method to richer formalisms of register automata, e.g., those over $\mathbb{R}$ with the less-than relation. Here, contradictions can still be recognized, since, e.g., a loop of strictly-less-than relations also implies non-satisfiability.

5.2 Semantic Synthesis

In this section, we will present the *Register Automaton Generator (RAGe)*¹¹ semantic synthesis framework for RAs. This algorithm was developed to support the evaluation of SL^{λ} shown in Section 4.3, which – as an active learning algorithm – does not benefit from structural synthesis. Instead, the evaluation requires automata accepting complex languages. To achieve this goal, the synthesis framework combines an established semantic synthesis technique for DFAs with the ability to enrich the generated automaton with structures intended to be difficult to learn.

¹¹<https://aqua.tools/rage>

The framework consists of three components, which we will introduce in the following sections. In [Section 5.2.1](#), we will introduce the synthesis algorithm that is used to create simple register automata of used-defined size. Next, in [Section 5.2.2](#), we will introduce a domain-specific language for defining RAs programmatically using a simple syntax. In [Section 5.2.3](#), we then discuss the combination operators that can be used to combine RAs, e.g., to introduce a programmatically-defined RA into a generated one. Finally, in [Section 5.2.4](#), we will draw a short conclusion and discuss how the approach can be applied for other types of EFSA.

This section is based on [Paper IV \[Die+24; Die+26a\]](#), however, the design of the synthesis framework was not described in that publication.

5.2.1 Synthesis Algorithm

Our synthesis approach is based on enhancing a suitable FSA with register features. By selecting a suitable synthesis algorithm for DFAs, we can retain its central properties in the final RA, but do not have to build our approach from the ground up. However, the register features we can add using this approach are limited.

We base our algorithm on the Champarnaud-Paranthoën algorithm that semantically synthesizes DFAs [\[CP05\]](#). This algorithm generates semantically unique DFAs by first creating a tree of states with unique, lexicographically smallest access strings, and then inserting other transitions that do not violate access string order. At three or more alphabet symbols, the languages created by the algorithm are almost certain to be unique unless almost all (or no) generated states are set to accepting.

We then post-process the generated DFAs¹² with input alphabet Σ into RAs without registers. We define a data universe $(\Lambda, D, \mathbf{a})$, a symbol mapping $m : \Sigma \rightarrow \Lambda$, and a used-defined guard $\hat{g}$. We then replace each DFA transition $\langle q, \sigma, q' \rangle$ with the RA transition $\langle q, m(\sigma), \hat{g}, \emptyset, q' \rangle$. While the result is a simple register automaton, the uniqueness of the DFA's language should translate to the RA, i.e., we are unlikely to generate RAs with identical languages during a synthesis campaign.

5.2.2 Domain-Specific Language

While the RAs synthesized using the approach described above are non-trivial, they do not exhibit the complex register operations that would be needed to cleanly differentiate the algorithms considered in the case study. However, it is relatively simple to define toy examples exhibiting these features. We therefore provide a domain-specific language to create such register automata.

¹²The actual implementation uses a modified algorithm that directly generates the specified RAs.

```

1  val xmpp = ra<String, String, InequalityGuard<String>> {
2      val x1 by register()
3      val x2 by register()
4
5      val register by symbolOfArity(2)
6      val login by symbolOfArity(2)
7      val pw by symbolOfArity(1)
8      val logout by symbol()
9      val delete by symbol()
10
11     val q2 by location()
12     val q3 by location(accepting=true)
13
14     transition(initial, register, True, assignments(x1 from register[0], x2 from
15         ↪ register[1]), q2)
16     transition(q2, login, (x1 eq login[0]) and (x2 eq login[1]), assignments(), q2)
17     transition(q3, logout, True, assignments(), q2)
18     transition(q3, pw, True, assignments(x2 from pw[0]), q3)
19     transition(q3, delete, True, assignments(), initial)
20 }

```

Listing 5.1: The XMPP register automaton defined via the RAGe DSL

For the DSL, we leverage the fact that the framework is implemented using the Kotlin programming language¹³, which offers excellent support for internal domain-specific languages. We use this to define a register automaton DSL that enables the programmatic creation of RAs while maintaining a high degree of readability. E.g., Listing 5.1 shows code equivalent to the XMPP register automaton in Figure 4.1. Since this is an internal DSL, a user can also mix static definitions with normal Kotlin language features. This enables the dynamic creation of repetitive features since, e.g., a long list of locations can be created using a for-loop.

5.2.3 Combination Operators

The third component enables the fusion of register automata using combination operators. This is intended to include toy-example sized, manually defined RAs that exhibit difficult-to-learn features into large, synthesized automata that can be scaled easily. However, the operators are capable of combining any two register automata, independent of their origin. We implement three combination operators: concatenation, partial replacement, and splitting by discriminator, each combining two register automata into one. The register sets of both automata are matched by name, so users can control the degree of shared registers. Examples

¹³<https://kotlinlang.org/>

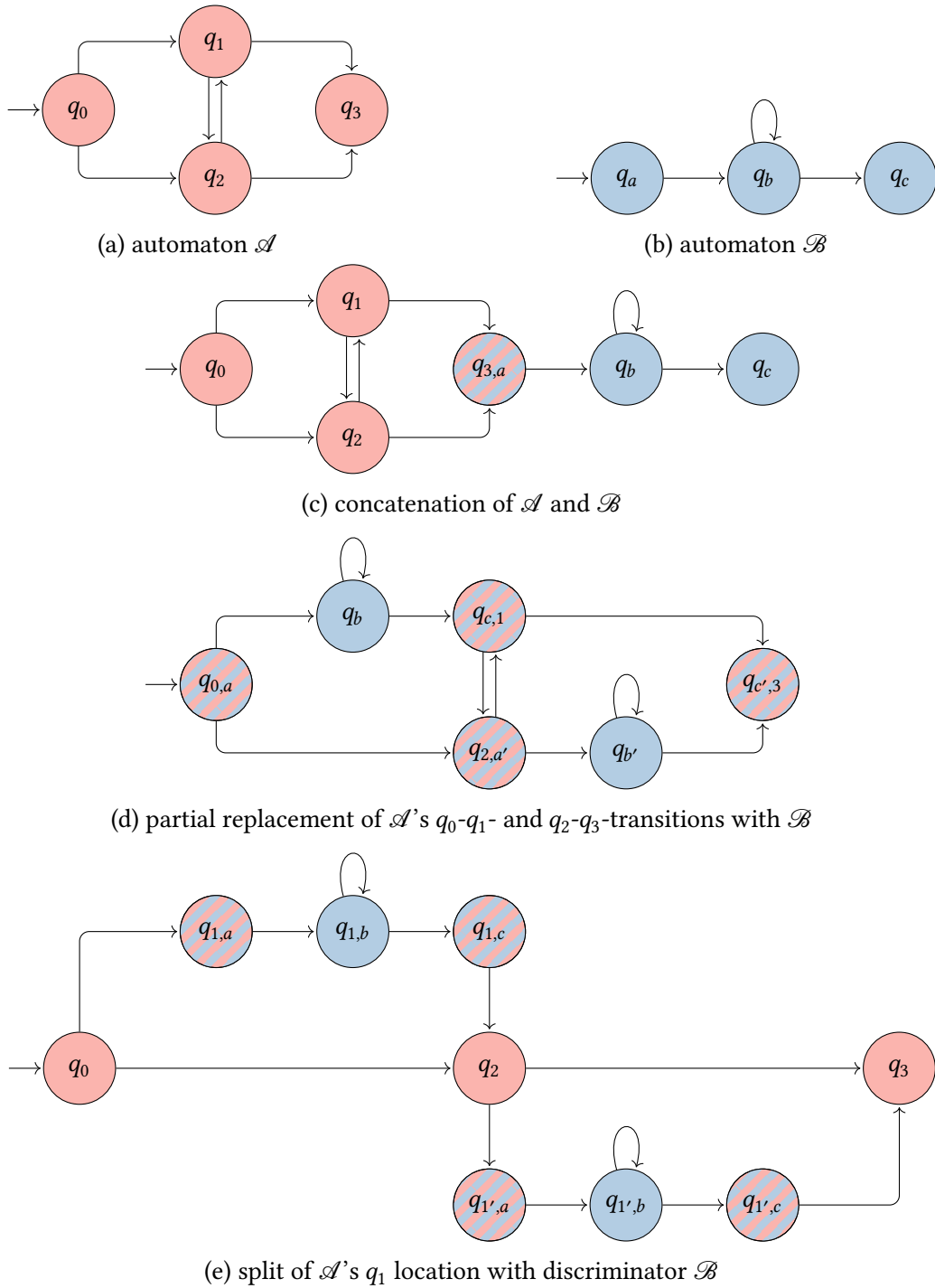


Figure 5.2: Examples of combinator application

for all operator applications are shown in Figure 5.2. In all cases, automata $\mathcal{A}$ and $\mathcal{B}$ (shown in Figures 5.2a and 5.2b, respectively) are combined. The examples omit transition labels for brevity.

Concatenation This operator places the second automaton “behind” the first automaton. To do this, it designates one of the locations most distant from the initial location of the first RA as the *most distant location*. It is then replaced with the initial location of the second RA. As a result, the automata are chained, but the distance heuristic forbids “shortcuts” around the first automaton.

An example of this is shown in Figure 5.2c: q_3 was selected as the most distant state and is replaced with q_a . The remainder of the automata is kept intact.

Partial Replacement This operator is intended to include the second automaton into the first multiple times by replacing transitions with the automaton. It randomly selects a defined share of independent transitions in the first RA. Transitions are independent if they share neither sources nor targets. Each transition is then replaced with a copy of the second RA, with the initial location corresponding to the transition’s source and its most distant location corresponding to the transition’s target. This can be used to insert a “difficult” example into a large, but “simple”, synthesized structure.

Figure 5.2d shows an example: one third of $\mathcal{A}$ ’s transitions are replaced. We have selected the transitions from q_0 to q_1 and from q_2 to q_3 for replacement. Each transition is replaced with a copy of $\mathcal{B}$, whose q_a and q_c locations are removed during the combination.

Splitting by Discriminator This operator is intended to facilitate the evaluation of active learners. It is designed to hinder the disambiguation of two locations, i.e., a learning algorithm can not easily find a disambiguating suffix. The operator selects a location in the first RA and splits it into four new locations: incoming left, incoming right, outgoing left, and outgoing right. The original location’s incoming and outgoing transitions are randomly distributed between the two new incoming and outgoing locations, respectively. As a result, the outgoing locations are (with high probability) *not* equivalent. Then, the second RA is inserted twice, once between both left and once between both right locations, using partial replacement. This means that the incoming states can only be disambiguated after exploring the second RA, which may be challenging to some MAT-style algorithms.

For an illustrative example, see Figure 5.2e. Here, the q_1 location has been selected for splitting. $q_{1,a}$ is the incoming left location and is assigned the q_0 - q_1 transition. $q_{1',a}$ is the incoming right location and obtains the q_2 - q_1 transition. $q_{1,c}$ and $q_{1',c}$ serve as outgoing left and right and are assigned the transitions to q_2 and q_3 , respectively. Next, the automaton $\mathcal{B}$ is inserted between each incoming-outgoing pair, merging them with its initial and most distant locations.

5.2.4 Conclusion

We have constructed a semantic synthesis algorithm for register automata by “piggybacking” on an existing approach for DFAs and introducing a mechanism for adding user-defined register automata using combination operators. Its application in Case Study II was successful, with the automata exhibiting the expected features and the DSL design enabling quick iterations on the generated models. The synthesis is fast; very large campaigns can be created in less than an hour.

Due to its current architecture, most components are interchangeable. The DFA synthesis algorithm can be exchanged for a different approach without compromising any other component. On the other hand, creating a different type of EFSA is also possible by defining a new enhancement mechanism, ensuring the combination operators work correctly (the difficulty of this is model-dependent), and optionally creating an appropriate DSL.

Using the approach of Champarnaud and Paranthoën to directly create RAs is likely not possible. A central part of the algorithm relies on building a tree of access strings (i.e., the lexicographically smallest inputs learning to a location). This requires the fan-out of each level to be fixed (for DFAs, it is the input alphabet size). However, for register automata, this is not the case for data parameters: on the first level, only a fresh value can be selected, on the second, the value can be repeated or a fresh value selected etc. This breaks a fundamental assumption of the algorithm. Admitting multiple parameters would address the issue, however, since the sequences p_1p_2 and p_2p_1 are functionally identical for a register automaton, this would cause redundancies in the generated automata.

Chapter 6

Recombinable Benchmarks and Tools

In [Chapter 4](#), we have established both the need for benchmark libraries as “living” entities and noticed that in benchmarking, works other than the state of the art are often not considered. Both are detrimental to the empirical content of results. Reasons for the phenomenon likely include hurdles in using benchmarks and issues with acquiring implementations of exiting algorithms. We propose the idea of *recombinable* benchmarks and tools that addresses both of these issues in tandem.

In this chapter, we will describe the concept of recombability in [Section 6.1](#). Then, we will sketch our proposed specification for recombable artifacts in [Section 6.2](#). In [Section 6.3](#), we will apply our approach to three sets of case studies. First, we will create recombable benchmarks for passive learning algorithms, second, we will build recombable tool and benchmark packages from the replication packaged constructed from the first three case studies in [Chapter 4](#), and third, we will discuss the “native” creation of recombable tools and benchmarks in the fourth case study (cf. [Section 4.5](#)). In [Section 6.4](#), we will summarize the results to estimate the effort required to adopt this approach.

This chapter is based on [Paper V \[DH25\]](#). For this thesis, we will focus on the application to EFSA benchmarks and have added the applications on Case Studies I–III.

6.1 Recombination

In current algorithmic research on FSAs and EFSAs, most evaluations are custom-crafted for the study at hand. Researchers collect a set of benchmarks they intend to use, convert them – if necessary – into a common format, and write an importer for the format. The algorithms that are benchmarked are implemented in software as a *tool*. Ideally, when benchmarking against a competitor, a software implementation compatible with the benchmarks is available. If not, the competitor must be re-implemented to enable a comparison.

We therefore propose the creation of separate, *recombinable* benchmark libraries and tool packages during research as shown in [Figure 6.1](#). In it, an evaluation of a passive learning

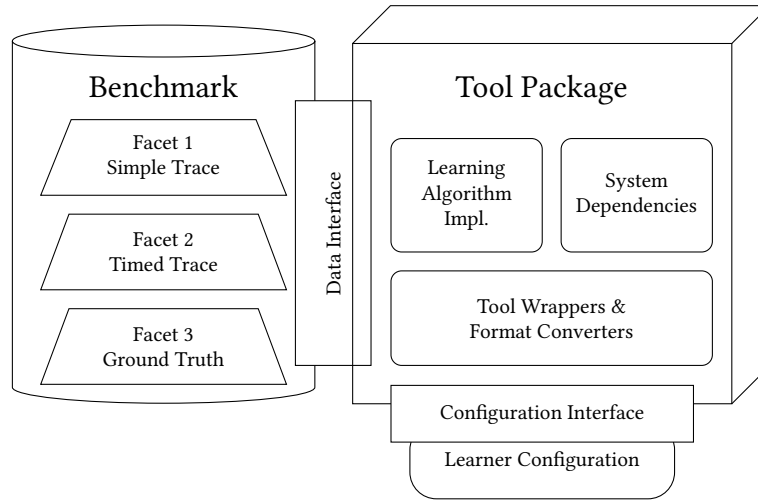


Figure 6.1: A recombinable learning tool and benchmark are integrated for an evaluation

algorithm using a benchmark library is shown. The algorithm is implemented in a tool package (i.e., a container) that contains the algorithm implementation, all necessary dependencies, and adapters for starting the implementation and converting data formats, if necessary. This ensures that no setup is required to use the tool package. The tool’s configuration is not “baked in”, but are loaded through an interface provided by the package. In a similar vein, the data interface is used for loading the benchmark itself and exfiltrating outputs.

Since we want to generate long-lived benchmarks that can be used in many studies, benchmark libraries need to aggregate as much information as possible. In the example, a benchmark might contain a ground truth automaton in addition to its traces. While the former may be useful in a subsequent evaluation step, it is irrelevant to the learner. Tools therefore must be able to ignore parts of a benchmark they do not care about. We therefore segment benchmark libraries into *facets*, each of which describes some form of data present in the benchmark.

Finally, we want to ensure compatibility between benchmarks and tools. We therefore need to describe each tool’s requirements w.r.t. the accepted benchmark format (e.g., an XML [WF04] document) and contents, i.e., the list of facets satisfied by it. Mirroring this, each benchmark must specify its format and the facets provided by it. Note that a compatible benchmark must exactly match the prescribed format, but can provide a superset of facets.

Such recombinable artifacts could then be combined “just in time” for an evaluation. Since setting up each combination requires little work, evaluations using this approach can be performed “at scale.” The more benchmarks and tools are included, the more exhaustive the evaluation can be, ultimately leading to a high empirical content. In addition, recombinable artifacts are suitable for long-term archival using a service such as Zenodo.¹⁴ By adding

¹⁴<https://zenodo.org/>

machine-readable descriptors for benchmark contents and tool requirements, we can enable the discovery of usable benchmark libraries and relevant competitors.¹⁵

6.2 Descriptors and Tool Format

We propose an *abstract* specification for three aspects of recombinable benchmarks. In it, we define a container format for tools, a description for benchmark contents and tool requirements, and a descriptor language for artifacts. We do intentionally *not* define a data format for benchmarks, since these are likely to be research community-specific. The specification therefore needs to be *instantiated* for each domain in which it is applied by defining formats and facets relevant for that domain. We will only outline the central aspects of the specification and refer to Paper V [DH25] for a more detailed presentation.

Our specification mandates the use of a single container format, namely Singularity¹⁶ [KSB17]. Singularity is an established container solution in high-performance computing and offers the “usual” benefits of containers, i.e., no need for setup or compilation of software. Singularity offers two advantages over other container ecosystems such as Docker. First, Singularity containers are designed to be shared as files, simplifying archiving and sharing. Second, Singularity enables direct access to the host file system, which addresses our need to load configuration files, load benchmarks, and extract result files.¹⁷

To describe a benchmark’s contents and a tool’s requirements (as well as possible outputs), we also utilize existing specifications. We describe a benchmark’s file format using media types¹⁸ [FB96]. If possible, we advocate the use of a structured data format such as JSON [Bra17] or XML [WF04]. This, in turn, enables us to rely on another common format to describe the facets offered or required by an artifact by describing them via JSON Schema [Wri+22]. Contrary to its name, it can describe most structured data formats and can define non-exhaustive schemata that ignore the presence of additional data. This enables a file to satisfy multiple schemata. We define one schema per facet and can thereby obtain a both machine-readable and machine-checkable description of a benchmark.

In some domains, established formats that do not allow additional data are used. For example, the Automata Wiki [Nei+19] RA format is an XML dialect, but it is not possible to attach additional data to an automaton. In these cases, we recommend only specifying a proprietary media type that mandates both file format and contents.

¹⁵Such benchmarks and tools meet both the *interoperable* and *reusable* principles of the FAIR [Wil+16] and FAIR4RS [Bar+22] criteria, respectively.

¹⁶<https://sylabs.io/singularity/>

¹⁷This is possible using other containers, but requires extra setup.

¹⁸Also commonly called MIME types.

Data Set	Features				Size	
	Classified	Normalized	TVT	Time	Logs	Entries
Abbadingo One	✓				440 000	8 390 011
ALFRED	✓				7 080	53 094
DISC	✓		✓		106 620	5 371 800
LogHub Hadoop	✓	✓		✓	978	179 994
MIT-AR	✓			✓	425	3 652

Table 6.1: Trace benchmarks libraries converted into a recombinable format

Both benchmarks and tools can then be described via descriptor files in YAML [YAM21] syntax. Benchmark descriptors describe the format and contents using media types and JSON Schema, while tool descriptors use the same notation for input and output formats. Both also contain metadata such as license information. Being plain text files, these descriptors can be archived alongside tool containers and benchmark files.

6.3 Application

We applied our specification in three scenarios. First, we instantiated our specification for benchmarks for passive automata learning and converted a set of established benchmark libraries to the new format. The results are described in Section 6.3.1. Second, we attempted to convert the replication packages from Case Studies I–III into recombinable artifacts and describe the results in Section 6.3.2. Third, we “natively” used our proposed format in Case Study IV to package both data and software tools. This is discussed in Section 6.3.3. For additional case studies in other domains, we refer to Paper V [DH25].

6.3.1 Trace Benchmarks

The first scenario is motivated by work in passive automata learning and is designed to measure the effort required to instantiate the specification, i.e., define a benchmark format and facets. We will describe the instantiation steps and measure the effort required for adapting existing benchmarks to the format by using lines of code as a proxy.

To benchmark passive learners, we only require a selection of traces that contain enough information for the learner to infer an automaton. An evaluation can then be performed by, e.g., splitting the traces into a training and validation set and learning an automaton on the former and measuring its performance on the latter. Since traces are also useful in a variety of other research fields and are, on occasion, produced as a side effect of other work, many

trace libraries exist that can be used as benchmarks. However, they all provide different types of data and are stored in different formats. We selected five libraries used in the literature.

Abbadingo One was an automata learning competition [LPP98] for which the training data is publicly available. Each trace defines a series of abstract input symbols and a classifier (accepted or rejected).

ALFRED is a benchmark for implementing domestic robots [Shr+20]. While it focuses on natural-language instructions and computer vision, it also contains machine-readable tasks and solution strategies. These can be used as traces to, e.g., learn Moore machines [Moo56] that recognize the robot’s task by the steps performed. Each trace then contains the list of steps, while the task can be used as a classifier.

DISC is a replication package by Shvo et al. that contains benchmarks for the referenced publication [Shv+21]. In addition to the trace data itself, this including the authors’ train-validate-test split.

LogHub Hadoop is a set of system logs from an Hadoop server [Lin+16] that was integrated into the LogHub trace collection [Zhu+23]. The logs are classified by the system’s mode of operation (i.e., did an error occur) and also contain the original timestamps for each event. Since this is a real system log, it contains arbitrary strings such as host names or IP addresses, however, we can use filters from the LogHub-2.0 data set [Jia+24] to discretize the logs.

MIT-AR is a collection of sensor data from two homes designed for activity recognition [TIL04]. The data contains trigger information for sensors installed in each apartment as well as information of the activities performed by its inhabitant (e.g., “watching TV”). The traces also contain time information.

From these libraries, we identified five facets that can be used to describe each library’s contents. An overview of the benchmarks and the facets satisfied by each is given in Table 6.1.

Base Trace The base facet of all traces. It mandates only the presence of a list of strings and can be combined with the remaining four facets.

Classification This facet requires that each trace is annotated with an additional *class* (e.g., the domestic tasks in the ALFRED and MIT-AR data sets). It is a prerequisite for learning Moore [Moo56] or Mealy [Mea55] machines. While it was present in each benchmark surveyed, this information is not required by, e.g., positive-example learners. We therefore did not include it in the base facet.

Normalization This facet was informed by register automata learning and the LogHub Hadoop traces. Since the log messages in this library are created from templates, they combine a (finite) set of templates with an (infinite) set of parameters for, e.g., host names. This transformation can be performed by normalization. The resulting trace

Data Set	Artifacts		Lines of Code			
	Benchmarks	Tools	Conversion	Packaging	Runtime	
Abbadingo One	16	0	37	:	:	0
ALFRED	1	0	75	:	:	0
DISC	120	0	53	68	(58)	0
LogHub Hadoop	2	0	101	:	:	0
MIT-AR	1	0	183	:	:	0
CS I: Backwards Propagation	64	2	0	134	(124)	20
CS II: SL ^λ	2 119	1	0	2 146	(27)	2
CS III: Intersection	0	1	0	17	(17)	0
CS IV: Heat Pump	1	2	35	38*	(38)*	0

*: not including packaging code shared with the benchmark sets

Table 6.2: Artifacts created and code required for recombability case studies

is defined over the (finite) template set, but each trace element additionally contains the parameters and the original value. Since this is an additional facet, use of the base value in FSA learning is also possible.

TVT-Splits Some data sets, especially those used in machine learning, define the training-validation-test split used for a study. Retaining this information is important to ensure reproducibility and perform comparable research. The facet mandates that each trace is annotated with its role: training, validation, or testing.

Timing This facet is motivated by the Hadoop and MIT-AR traces and is relevant for timed automata learning. We specified the facet loosely to model both data sets, since they contain different amounts of information. The facet mandates that each trace element contains a start time relative to the trace’s start. Optionally, a trace can also define relative end times for each entry, a relative end time for the trace, and an *absolute* start date. This can, e.g., model cases like “three seconds after boot”, but also describe data with precisely known absolute start and end dates.

Next, we formalized these facets as JSON schemata and decided on (compressed) JSON as the basic data format due to its wide adoption for storing structured data. Although JSON is not primarily designed for human-readability, we did not consider this to be essential for benchmark data containing millions of entries. We then developed a conversion framework in Kotlin that transforms each benchmark’s original format to our proposed JSON-based format and creating packaging scripts that handle the fully automatic conversion of the original files into our format.

An overview over the effort required is shown in the upper block of [Table 6.2](#). Very little code is necessary for both conversion and packaging. Note that packaging code is shared between all five benchmarks; the number in brackets indicates lines of code without the enumeration of all benchmark names. Anecdotaly, development of all converters took a few hours at most, including time spent for looking up documentation, integrating parser libraries etc. The most complex converter is the MIT-AR converter, mostly due to the complexity of reading the benchmark's Matlab file format.

6.3.2 Case Study I–III Replication Packages

The second scenario retroactively applies the approach to Case Studies I–III (cf. [Sections 4.2](#) to [4.4](#)). Here, we do not instantiate a benchmark format, but instead split each case study's replication package into tool(s) and benchmark set to measure the effort required in converting existing bundles. We did not utilize source code or similar artifacts.

Case Study I: Backwards Propagation Here, a replication package in the form of a Docker container [[DH22b](#)] was provided. We created two tools, our backwards-propagating converter and the baseline RALib-based forwards-propagating model checker, and one benchmark set containing all automata used for benchmarking.

Conversion was hampered by the fact that the backwards-propagating tool expects to be bundled with its benchmarks. We had to adapt its startup script to load benchmarks from an externally-accessible path and write a wrapper that copies user-provided automata to said path and renames them to the expected format. For the RALib-based tool, an existing wrapper script could be adapted with less effort.

Case Study II: SL^λ For this case study, the replication package [[Sag+23](#)] used the format mandated by TACAS, i.e., a ZIP archive designed to be used on a specific virtual machine. Since all three learning algorithms are bundled in a single application, we created a single tool. Additionally, we constructed a large benchmark set of all automata present in the replication package.

This conversion was relatively simple, since the learner already provided facilities for building a standalone application that just requires a Java runtime. We could install it into a vanilla JVM container and added a trivial wrapper script.

Case Study III: Intersection This case study also used a Docker-based replication package [[Die+23b](#)]. Since each benchmark requires programmatic setup, it is impossible to separate benchmarks and tools in it. We therefore only repackaged the full replication package into a tool container that does not accept any inputs.

For the first two case studies, even conversion from a format not designed for recombination was relatively simple. An overview is shown in the middle block of [Table 6.2](#), with very little

code being required. Note that the packaging code needs to enumerate every benchmark, inflating the lines of code. Therefore, the number in brackets is a more accurate representation of the real effort. Anecdotally, each package required only a few hours of work. The stronger the integration between tool and benchmark, the more code was necessary, with the backwards propagating tool requiring the most effort.

The third case study showcases the approach’s inevitable shortcomings for use-case specific applications. Here, it is not possible to easily separate tools and benchmarks. An architecture more amenable to recombination could split the learner into separate trace database components and the remaining application. The trace databases could then be bundled with the benchmark, although it is questionable if they can meaningfully be used in other work.

6.3.3 Case Study IV Tools and Data

In the third scenario, we applied our format during the development of the learning pipeline in Case Study IV (cf. [Section 4.5](#)) to measure the overhead incurred when using the proposed format *during* a study. The case study’s work was conducted in collaboration with another team. While we implemented the select & merge and learning steps, our collaborators implemented the discretizer and the metaheuristic optimization framework. The pipeline uses two data formats: a CSV-based format used by the discretizer and our proposed trace format for our tools. In addition to the converters described in [Section 6.3.1](#), we therefore developed a conversion tool for the CSV-based format. Due to it being completed last, we did not package the metaheuristic.

Since our steps were designed to “natively” support our proposed tool format in their build system, the packaging code only needs to invoke its build script. We additionally implemented packaging for the discretizer based on its source code distribution. However, this was also simple to implement, requiring only few lines of code. Finally, data format conversion for the CSV-based format was easier to implement than that for the established benchmarks, requiring less lines of code than any of these. A summary is shown in the final block of [Table 6.2](#).

This experiment showcases the benefits of constructing tools and benchmarks for recombination during development. Conversions into the proposed format were simpler than the retroactive conversions for Case Studies I–III. Since each step in the pipeline can be packaged as either a separate tool or invoked in separation from a combined tool executable, the resulting recombinable tools are more versatile than those built from replication packages.

6.4 Summary

In conclusion, our approach can be applied to the domain of EFSA algorithm research with reasonable effort. While adaption of existing tools and benchmarks is not entirely trivial, we were able to adopt five benchmark libraries and two of three replication packages to a matching format with limited overhead. Only bespoke algorithms designed for a single case study could not be meaningfully represented using this approach. “Native” use of the proposed format seems even more promising, with very little adaptation overhead required.

Looking to the future, adoption of this approach may address two of the issues we identified concerning the current state of EFSA benchmarking. First, the creation and use of recombining tool packages enables evaluations comparing a novel algorithm against a large set of competitors instead of only a few or one. Second, building recombining, “living” benchmark libraries that can be used with such tools may avoid deprecation of benchmark libraries, incentivize their use in studies, and support the creation of more and larger EFSA benchmark libraries.

Chapter 7

Summary

In this chapter, we will first summarize the contributions made by this thesis for each of the four research questions and highlight key results in [Section 7.1](#). Then, we will offer an outlook on possible future work in [Section 7.2](#).

7.1 Conclusion

In this thesis, we have tackled the problem of benchmark selection in algorithms research for extended finite state automata. To this end, we have created a taxonomy of benchmarks that can be used to describe categories of benchmark used in empirical evaluations. We then established limits in the availability and use of benchmarks and identified their causes. Finally, we improved upon the current state of the art in benchmarking in both synthesis algorithms and benchmark formats. We will now discuss the contributions in detail.

We have shown that register automaton benchmarks, such as those contained in Automata Wiki benchmark library [[Nei+19](#)], can not feasibly be used to perform benchmarks in other register automaton formalisms. Converting into almost all more restrictive formalisms causes exponential size increases or, in one case, loss of determinism. Removing restrictions is possible, but models in the most restrictive formalism are far larger than this in less restrictive ones. This shows the limitations of libraries in EFSA research, where conversions between formalisms is usually difficult or impossible.

By surveying existing world and applying our benchmark taxonomy, we have obtained an overview of the evolution of benchmarking approaches in EFSA and FSA research. We have observed a natural progression from theoretical works over works benchmarking on toy examples to benchmarking using case studies. We also found evidence of three challenges in benchmarking that are addressed in the thesis. For benchmark libraries, we have observed a life cycle of adoption, in which libraries eventually stop being used. Another common issue in benchmarking is the limited number of comparators used for evaluation, with many publications only benchmarking against a single competitor. Comparing FSA and EFSA

research, can observe that benchmark libraries and synthesis have become established in FSA, but not EFSA research.

Using four case studies with different approaches to benchmarking and different empirical content, we have established that synthetic and real world-inspired benchmarks complement each other. Synthetic benchmarks excel at demonstrating scaling effects on algorithms by allowing studies in which a specific property, e.g., automaton size, is set to arbitrary levels. However, they usually lack realism, which can be provided using real-world inspired benchmarks. Using only these, however, cannot establish scalability. By combining both approaches, we can support statements with a high empirical content.

The same case studies also demonstrate that in general, case studies and toy examples do not serve to support conclusions with high empirical content. A research project using both could only make weak conjectures about the algorithm's performance on unknown models, while benchmark libraries allow for stronger conclusions. However, we also encountered the issue of "outrunning" a benchmark library (i.e., the baseline comparator could already solve it satisfactorily), which shows the need to grow benchmark libraries with the state of the art.

We have described a structural synthesis approach for register automata that can avoid the creation of dead transitions and unreachable locations, which remaining highly configurable. We successfully applied the algorithm to support one of the case studies. This lays groundwork for structural synthesis of register automata and other EFSA's operating on guards expressed in propositional logic.

We also define a semantic synthesis framework for register automata that extends the DFA synthesis algorithm of Champarnaud and Paranthoën [CP05]. The framework enables combination of synthetic automata with toy examples to ensure semantic properties of the resulting automata and was successfully used in a case study. This is the first algorithm to enable semantic synthesis of EFSA's with storage.

Finally, we have introduced the notion of recombinable tools and benchmarks and defined a specification for the creation of these. Adoption of such techniques simplifies the use of benchmarks and the use of algorithm implementations as comparators and may incentivize the creation and maintenance of long-lived benchmark libraries, both in EFSA research and adjacent fields.

In summary, we have identified the combination of well-maintained benchmark libraries and synthetic automata as the "gold standard" of algorithm research on EFSA's. We were able to both identify and address key inhibitors to the adoption of this approach, both in the research forming part of this thesis and future studies.

7.2 Future Work

Many of the contributions made in this thesis give rise to further questions and lines of work. We will outline these in the sequence of contributions used in the [previous section](#).

While we have considered transformations between a large portion of register automata present in the literature, the results do not cover extended types of register automata such as pushdown RAs [MRT17] or other types of automata for words over infinite alphabets such as data automata [Boj+06]. An investigation into conversions between those would provide insight into both the models and the benchmarking landscape for these. Beyond these, transformations for completely different types of EFSA, e.g., timed automata [AD94], are also important to understand both the models and the potential for reusing benchmarks.

In this work, our taxonomy of benchmarks has only been applied to an informal survey and four case studies. In future work, both a structured literature survey and more case studies could sharpen the insights gained in this thesis and allow for a structured theory of benchmarking in Software Engineering research. Also, our survey hinted at patterns of reuse of benchmarks and competitor choice in evaluations, with some publications intentionally reusing benchmarks from competitors. An investigation into the “genealogy” of benchmarks might yield new insights into research techniques.

This thesis has made first inroads into both structural and semantic synthesis of EFSA benchmarks. However, both algorithms are limited to generating register automata; in addition, the semantic synthesizer is hampered by the transformation from the underlying DFA synthesizer. Open questions include a “native” semantic synthesis approach for RAs, as well as synthesis of other EFSA models (e.g., those mentioned above).

While our proposed recombability standard is intended to be universally usable, it has so far only been employed in the works contributing to this thesis. More case studies on its use are needed to identify its strengths and weaknesses. In addition, our specification for tool packages could be extended to cover more complex use cases, such as tool pipelines or repeated invocation of the same tool (e.g., an oracle for learning). In a similar vein, fully formalizing interactions with tools (e.g., how to pass CLI parameters) could reduce the effort required to set up an evaluation to virtually zero. Both proposals are likely to increase adoption of the format, indirectly contributing to benchmark availability.

Bibliography

- [Aar+12] Fides Aarts, Faranak Heidarian, Harco Kuppens, Petur Olsen, and Frits Vaandrager. “Automata Learning through Counterexample Guided Abstraction Refinement.” In: *FM 2012: Formal Methods*. Ed. by Dimitra Giannakopoulou and Dominique Méry. Lecture Notes in Computer Science 7436. Berlin, Heidelberg: Springer, 2012, pp. 10–27. ISBN: 978-3-642-32759-9. DOI: [10.1007/978-3-642-32759-9_4](https://doi.org/10.1007/978-3-642-32759-9_4) (cit. on p. 47).
- [Aar+15] Fides Aarts, Paul Fiterau-Brostean, Harco Kuppens, and Frits Vaandrager. “Learning Register Automata with Fresh Value Generation.” In: *Theoretical Aspects of Computing - ICTAC 2015*. Ed. by Martin Leucker, Camilo Rueda, and Frank D. Valencia. Lecture Notes in Computer Science 9399. Cham: Springer International Publishing, 2015, pp. 165–183. ISBN: 978-3-319-25150-9. DOI: [10.1007/978-3-319-25150-9_11](https://doi.org/10.1007/978-3-319-25150-9_11) (cit. on p. 47).
- [ABL02] Glenn Ammons, Rastislav Bodík, and James R. Larus. “Mining Specifications.” In: *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. New York, NY, USA: Association for Computing Machinery, Jan. 2002, pp. 4–16. ISBN: 978-1-58113-450-6. DOI: [10.1145/503272.503275](https://doi.org/10.1145/503272.503275) (cit. on p. 33).
- [AD01] Imtiaz Ahmad and A. Shoba Das. “A Heuristic Algorithm for the Minimization of Incompletely Specified Finite State Machines.” In: *Computers & Electrical Engineering* 27.2 (May 2001), pp. 159–172. ISSN: 0045-7906. DOI: [10.1016/S0045-7906\(00\)00016-1](https://doi.org/10.1016/S0045-7906(00)00016-1) (cit. on p. 37).
- [AD94] Rajeev Alur and David L. Dill. “A Theory of Timed Automata.” In: *Theoretical Computer Science* 126.2 (Apr. 1994), pp. 183–235. ISSN: 0304-3975. DOI: [10.1016/0304-3975\(94\)90010-8](https://doi.org/10.1016/0304-3975(94)90010-8) (cit. on pp. 28, 101).
- [ADP13] Fides Aarts, Joeri De Ruiter, and Erik Poll. “Formal Models of Bank Cards for Free.” In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. Institute of Electrical and Electronics Engineers, Mar. 2013, pp. 461–468. ISBN: 978-0-7695-4993-4. DOI: [10.1109/ICSTW.2013.60](https://doi.org/10.1109/ICSTW.2013.60) (cit. on p. 44).
- [AEF15] Dana Angluin, Sarah Eisenstat, and Dana Fisman. “Learning Regular Languages via Alternating Automata.” In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*. Ed. by Qiang Yang and Michael Wooldridge. Palo Alto, California, USA: AAAI Press, 2015, pp. 3308–3314. ISBN: 978-1-57735-738-4. URL: <https://www.ijcai.org/Abstract/15/466> (cit. on p. 40).

- [AH90] James Aspnes and Maurice Herlihy. “Fast Randomized Consensus Using Shared Memory.” In: *Journal of Algorithms* 11.3 (Sept. 1990), pp. 441–461. ISSN: 0196-6774. DOI: [10.1016/0196-6774\(90\)90021-6](https://doi.org/10.1016/0196-6774(90)90021-6) (cit. on pp. 33, 34).
- [AJ06] Pieter Adriaans and Ceriël Jacobs. “Using MDL for Grammar Induction.” In: *Grammatical Inference: Algorithms and Applications*. Ed. by Yasubumi Sakakibara, Satoshi Kobayashi, Kengo Sato, Tetsuro Nishino, and Etsuji Tomita. Lecture Notes in Computer Science 4201. Berlin, Heidelberg: Springer, 2006, pp. 293–306. ISBN: 978-3-540-45265-2. DOI: [10.1007/11872436_24](https://doi.org/10.1007/11872436_24) (cit. on p. 37).
- [AJU10] Fides Aarts, Bengt Jonsson, and Johan Uijen. “Generating Models of Infinite-State Communication Protocols Using Regular Inference with Abstraction.” In: *Testing Software and Systems*. Ed. by Alexandre Petrenko, Adenilso Simão, and José Carlos Maldonado. Lecture Notes in Computer Science 6435. Berlin, Heidelberg: Springer, 2010, pp. 188–204. ISBN: 978-3-642-16573-3. DOI: [10.1007/978-3-642-16573-3_14](https://doi.org/10.1007/978-3-642-16573-3_14) (cit. on pp. 44, 45, 49, 55).
- [Ami+15] Francesco Amigoni, Emanuele Bastianelli, Jakob Berghofer, Andrea Bonarini, Giulio Fontana, Nico Hochgeschwender, Luca Iocchi, Gerhard Kraetzschmar, Pedro Lima, Matteo Matteucci, Pedro Miraldo, Daniele Nardi, and Viola Schiaffonati. “Competitions for Benchmarking: Task and Functionality Scoring Complete Performance Assessment.” In: *IEEE Robotics & Automation Magazine* 22.3 (Sept. 2015), pp. 53–61. ISSN: 1558-223X. DOI: [10.1109/MRA.2015.2448871](https://doi.org/10.1109/MRA.2015.2448871) (cit. on p. 12).
- [AMN94] Helena Ahonen, Heikki Mannila, and Erja Nikunen. “Forming Grammars for Structured Documents: An Application of Grammatical Inference.” In: *Grammatical Inference and Applications*. Ed. by Rafael C. Carrasco and Jose Oncina. Lecture Notes in Computer Science 862. Berlin, Heidelberg: Springer, 1994, pp. 153–167. ISBN: 978-3-540-48985-6. DOI: [10.1007/3-540-58473-0_145](https://doi.org/10.1007/3-540-58473-0_145) (cit. on p. 33).
- [Amp+19] Elvio Amparore, Bernard Berthomieu, Gianfranco Ciardo, Silvano Dal Zilio, Francesco Gallà, Lom Messan Hillah, Francis Hulin-Hubard, Peter Gjørl Jensen, Loïc Jezequel, Fabrice Kordon, Didier Le Botlan, Torsten Liebke, Jeroen Meijer, Andrew Miner, Emmanuel Paviot-Adet, Jiří Srba, Yann Thierry-Mieg, Tom van Dijk, and Karsten Wolf. “Presentation of the 9th Edition of the Model Checking Contest.” In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Dirk Beyer, Marieke Huisman, Fabrice Kordon, and Bernhard Steffen. Lecture Notes in Computer Science 11429. Cham: Springer International Publishing, 2019, pp. 50–68. ISBN: 978-3-030-17502-3. DOI: [10.1007/978-3-030-17502-3_4](https://doi.org/10.1007/978-3-030-17502-3_4) (cit. on p. 38).
- [AMP22] Bernhard K. Aichernig, Edi Muškardin, and Andrea Pferscher. “Active Vs. Passive: A Comparison of Automata Learning Paradigms for Network Protocols.” In: *Proceedings Fourth International Workshop on Formal Methods for Autonomous Systems (FMAS) and Fourth International Workshop on Automated and Verifiable Software System Development (ASYDE)*. Ed. by Matt Luckcuck and Marie Farrell. Electronic Proceedings in Theoretical Computer Science 371. Waterloo, NSW, Australia: Open Publishing Association, Sept. 2022, pp. 1–19. DOI: [10.4204/EPTCS.371.1](https://doi.org/10.4204/EPTCS.371.1) (cit. on p. 34).
- [AMR07] Marco Almeida, Nelma Moreira, and Rogério Reis. “Enumeration and Generation with a String Automata Representation.” In: *Theoretical Computer Science* 387.2 (Nov. 2007), pp. 93–102. ISSN: 0304-3975. DOI: [10.1016/j.tcs.2007.07.029](https://doi.org/10.1016/j.tcs.2007.07.029) (cit. on p. 14).

- [An+20] Jie An, Mingshuai Chen, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. “Learning One-Clock Timed Automata.” In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Armin Biere and David Parker. Lecture Notes in Computer Science 12078. Cham: Springer International Publishing, 2020, pp. 444–462. ISBN: 978-3-030-45190-5. DOI: [10.1007/978-3-030-45190-5_25](https://doi.org/10.1007/978-3-030-45190-5_25) (cit. on pp. 44, 48).
- [An+21] Jie An, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. “Learning Nondeterministic Real-Time Automata.” In: *ACM Transactions on Embedded Computing Systems* 20.5, 99 (Sept. 2021). ISSN: 1539-9087. DOI: [10.1145/3477030](https://doi.org/10.1145/3477030) (cit. on pp. 45, 46, 48).
- [Ang87] Dana Angluin. “Learning Regular Sets from Queries and Counterexamples.” In: *Information and Computation* 75.2 (Nov. 1987), pp. 87–106. ISSN: 0890-5401. DOI: [10.1016/0890-5401\(87\)90052-6](https://doi.org/10.1016/0890-5401(87)90052-6) (cit. on pp. 1, 13, 30, 32, 55, 56, 65).
- [AP18] Florent Avellaneda and Alexandre Petrenko. “FSM Inference from Long Traces.” In: *Formal Methods*. Ed. by Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik de Vink. Lecture Notes in Computer Science 10951. Cham: Springer International Publishing, 2018, pp. 93–109. ISBN: 978-3-319-95582-7. DOI: [10.1007/978-3-319-95582-7_6](https://doi.org/10.1007/978-3-319-95582-7_6) (cit. on pp. 33, 39).
- [APT20] Bernhard K. Aichernig, Andrea Pferscher, and Martin Tappler. “From Passive to Active: Learning Timed Automata Efficiently.” In: *NASA Formal Methods*. Ed. by Ritchie Lee, Susmit Jha, Anastasia Mavridou, and Dimitra Giannakopoulou. Lecture Notes in Computer Science 12229. Cham: Springer International Publishing, 2020, pp. 1–19. ISBN: 978-3-030-55754-6. DOI: [10.1007/978-3-030-55754-6_1](https://doi.org/10.1007/978-3-030-55754-6_1) (cit. on pp. 45, 48).
- [AR15] Andreas Abel and Jan Reineke. “MeMin: SAT-Based Exact Minimization of Incompletely Specified Mealy Machines.” In: *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Institute of Electrical and Electronics Engineers, Nov. 2015, pp. 94–101. ISBN: 978-1-4673-8388-2. DOI: [10.1109/ICCAD.2015.7372555](https://doi.org/10.1109/ICCAD.2015.7372555) (cit. on p. 37).
- [AS13] Alex D. B. Alberto and Adenilso Simao. “Iterative Minimization of Partial Finite State Machines.” In: *Central European Journal of Computer Science* 3.2 (June 2013), pp. 91–103. ISSN: 2081-9935. DOI: [10.2478/s13537-013-0106-0](https://doi.org/10.2478/s13537-013-0106-0) (cit. on p. 37).
- [AS83] Dana Angluin and Carl H. Smith. “Inductive Inference: Theory and Methods.” In: *ACM Computing Surveys* 15.3 (Sept. 1983), pp. 237–269. ISSN: 0360-0300. DOI: [10.1145/356914.356918](https://doi.org/10.1145/356914.356918) (cit. on p. 13).
- [ASV10] Fides Aarts, Julien Schmaltz, and Frits Vaandrager. “Inference and Abstraction of the Biometric Passport.” In: *Leveraging Applications of Formal Methods, Verification, and Validation*. Ed. by Tiziana Margaria and Bernhard Steffen. Lecture Notes in Computer Science 6415. Berlin, Heidelberg: Springer, 2010, pp. 673–686. ISBN: 978-3-642-16558-0. DOI: [10.1007/978-3-642-16558-0_54](https://doi.org/10.1007/978-3-642-16558-0_54) (cit. on pp. 4, 7, 44).
- [ASZ21] Shahbaz Ali, Hailong Sun, and Yongwang Zhao. “Model Learning: A Survey of Foundations, Tools and Applications.” In: *Frontiers of Computer Science* 15.5, 155210 (June 2021). ISSN: 2095-2236. DOI: [10.1007/s11704-019-9212-z](https://doi.org/10.1007/s11704-019-9212-z) (cit. on p. 13).
- [AT19] Bernhard K. Aichernig and Martin Tappler. “Probabilistic Black-Box Reachability Checking (Extended Version).” In: *Formal Methods in System Design* 54.3 (Nov. 2019), pp. 416–448. ISSN: 1572-8102. DOI: [10.1007/s10703-019-00333-0](https://doi.org/10.1007/s10703-019-00333-0) (cit. on pp. 33, 35).

- [ATW24] Bernhard K. Aichernig, Martin Tappler, and Felix Wallner. “Benchmarking Combinations of Learning and Testing Algorithms for Automata Learning.” In: *Formal Aspects of Computing* 36.1, 3 (Mar. 2024). ISSN: 0934-5043. DOI: [10.1145/3605360](https://doi.org/10.1145/3605360) (cit. on pp. 12, 38, 76).
- [AV10] Fides Aarts and Frits Vaandrager. “Learning I/O Automata.” In: *CONCUR 2010 - Concurrency Theory*. Ed. by Paul Gastin and François Laroussinie. Lecture Notes in Computer Science 6269. Berlin, Heidelberg: Springer, 2010, pp. 71–85. ISBN: 978-3-642-15375-4. DOI: [10.1007/978-3-642-15375-4_6](https://doi.org/10.1007/978-3-642-15375-4_6) (cit. on pp. 34, 36).
- [Bai+03] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. “Model-Checking Algorithms for Continuous-Time Markov Chains.” In: *IEEE Transactions on Software Engineering* 29.6 (June 2003), pp. 524–541. ISSN: 1939-3520. DOI: [10.1109/TSE.2003.1205180](https://doi.org/10.1109/TSE.2003.1205180) (cit. on p. 73).
- [Bal+97] José L. Balcázar, Josep Díaz, Ricard Gavaldà, and Osamu Watanabe. “Algorithms for Learning Finite Automata from Queries: A Unified View.” In: *Advances in Algorithms, Languages, and Complexity*. Ed. by Ding-Zhu Du and Ker-I. Ko. Boston, MA: Springer US, 1997, pp. 53–72. ISBN: 978-1-4613-3394-4. DOI: [10.1007/978-1-4613-3394-4_2](https://doi.org/10.1007/978-1-4613-3394-4_2) (cit. on p. 13).
- [Bar+22] Michelle Barker, Neil P. Chue Hong, Daniel S. Katz, Anna-Lena Lamprecht, Carlos Martinez-Ortiz, Fotis Psomopoulos, Jennifer Harrow, Leyla Jael Castro, Morane Gruenpeter, Paula Andrea Martinez, and Tom Honeyman. “Introducing the FAIR Principles for Research Software.” In: *Scientific Data* 9.1 (Oct. 2022), p. 622. ISSN: 2052-4463. DOI: [10.1038/s41597-022-01710-x](https://doi.org/10.1038/s41597-022-01710-x) (cit. on p. 91).
- [BDN08] Frédérique Bassino, Julien David, and Cyril Nicaud. “Enumeration and Random Generation of Possibly Incomplete Deterministic Automata.” In: *Pure Mathematics and Applications* 19.2 (2008), pp. 1–16 (cit. on p. 14).
- [Ber+19] Mario Luca Bernardi, Marta Cimitile, Damiano Distanti, Fabio Martinelli, and Francesco Mercaldo. “Dynamic Malware Detection and Phylogeny Analysis Using Process Mining.” In: *International Journal of Information Security* 18.3 (June 2019), pp. 257–284. ISSN: 1615-5270. DOI: [10.1007/s10207-018-0415-3](https://doi.org/10.1007/s10207-018-0415-3) (cit. on p. 37).
- [Bes+11] Ivan Beschastnikh, Yuriy Brun, Sigurd Schneider, Michael Sloan, and Michael D. Ernst. “Leveraging Existing Instrumentation to Automatically Infer Invariant-Constrained Models.” In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, Sept. 2011, pp. 267–277. ISBN: 978-1-4503-0443-6. DOI: [10.1145/2025113.2025151](https://doi.org/10.1145/2025113.2025151) (cit. on p. 43).
- [BF72] A. W. Biermann and J. A. Feldman. “On the Synthesis of Finite-State Machines from Samples of Their Behavior.” In: *IEEE Transactions on Computers* C-21.6 (1972), pp. 592–597. ISSN: 1557-9956. DOI: [10.1109/TC.1972.5009015](https://doi.org/10.1109/TC.1972.5009015) (cit. on pp. 1, 32, 65, 71).
- [BFL04] Sébastien Bardin, Alain Finkel, and Jérôme Leroux. “FASTer Acceleration of Counter Automata in Practice.” In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Kurt Jensen and Andreas Podelski. Lecture Notes in Computer Science 2988. Berlin, Heidelberg: Springer, 2004, pp. 576–590. ISBN: 978-3-540-24730-2. DOI: [10.1007/978-3-540-24730-2_42](https://doi.org/10.1007/978-3-540-24730-2_42) (cit. on p. 38).

- [BFT25] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Smt-Lib Standard. Version 2.7*. Release: July 7, 2025. July 7, 2025. URL: <https://smt-lib.org/> (cit. on p. 14).
- [BJR06] Therese Berg, Bengt Jonsson, and Harald Raffelt. “Regular Inference for State Machines with Parameters.” In: *Fundamental Approaches to Software Engineering*. Ed. by Luciano Baresi and Reiko Heckel. Lecture Notes in Computer Science 3922. Berlin, Heidelberg: Springer, 2006, pp. 107–121. ISBN: 978-3-540-33094-3. DOI: [10.1007/11693017_10](https://doi.org/10.1007/11693017_10) (cit. on p. 44).
- [Bli+16] Thomas Bligaard, R. Morris Bullock, Charles T. Campbell, Jingguang G. Chen, Bruce C. Gates, Raymond J. Gorte, Christopher W. Jones, William D. Jones, John R. Kitchin, and Susannah L. Scott. “Toward Benchmarking in Catalysis Science: Best Practices, Challenges, and Opportunities.” In: *ACS Catalysis* 6.4 (Apr. 2016), pp. 2590–2602. DOI: [10.1021/acscatal.6b00183](https://doi.org/10.1021/acscatal.6b00183) (cit. on p. 12).
- [BM18] Amel Bennaceur and Karl Meinke. “Machine Learning for Software Analysis.” In: *Machine Learning for Dynamic Software Analysis: Potentials and Limits. Models, Methods, and Applications*. Ed. by Amel Bennaceur, Reiner Hähnle, and Karl Meinke. Lecture Notes in Computer Science 11026. Cham: Springer International Publishing, 2018, pp. 3–49. ISBN: 978-3-319-96562-8. DOI: [10.1007/978-3-319-96562-8_1](https://doi.org/10.1007/978-3-319-96562-8_1) (cit. on p. 13).
- [BN07] Frédérique Bassino and Cyril Nicaud. “Enumeration and Random Generation of Accessible Automata.” In: *Theoretical Computer Science* 381.1 (Aug. 2007), pp. 86–104. ISSN: 0304-3975. DOI: [10.1016/j.tcs.2007.04.001](https://doi.org/10.1016/j.tcs.2007.04.001) (cit. on p. 14).
- [Boj+06] Mikołaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. “Two-Variable Logic on Words with Data.” In: *21st Annual IEEE Symposium on Logic in Computer Science (LICS’06)*. Institute of Electrical and Electronics Engineers, Aug. 2006, pp. 7–16. ISBN: 0-7695-2631-4. DOI: [10.1109/LICS.2006.51](https://doi.org/10.1109/LICS.2006.51) (cit. on p. 101).
- [Bol+09] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. “Angluin-Style Learning of NFA.” In: *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2009, pp. 1004–1009. ISBN: 978-1-57735-426-0. URL: <https://www.ijcai.org/Abstract/09/170> (cit. on pp. 40, 41).
- [Bra+99] Volker Braun, Jürgen Kreidler, Tiziana Margaria, and Bernhard Steffen. “The ETI Online Service in Action.” In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by W. Rance Cleaveland. Berlin, Heidelberg: Springer, 1999, pp. 439–443. ISBN: 978-3-540-49059-3. DOI: [10.1007/3-540-49059-0_31](https://doi.org/10.1007/3-540-49059-0_31) (cit. on p. 14).
- [Bra17] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. Internet Standard 90. RFC Editor, Dec. 2017. DOI: [10.17487/RFC8259](https://doi.org/10.17487/RFC8259) (cit. on p. 91).
- [Bro+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. arXiv Preprint arXiv:1606.01540. June 2016. DOI: [10.48550/arXiv.1606.01540](https://doi.org/10.48550/arXiv.1606.01540) (cit. on p. 38).
- [BS25] Dirk Beyer and Jan Strejček. “Improvements in Software Verification and Witness Validation: SV-COMP 2025.” In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Arie Gurfinkel and Marijn Heule. Lecture Notes in Computer Science

15698. Cham: Springer Nature Switzerland, 2025, pp. 151–186. ISBN: 978-3-031-90660-2. DOI: [10.1007/978-3-031-90660-2_9](https://doi.org/10.1007/978-3-031-90660-2_9) (cit. on p. 14).
- [BSH22] Alexander Bainczyk, Bernhard Steffen, and Falk Howar. “Lifelong Learning of Reactive Systems in Practice.” In: *The Logic of Software. A Tasting Menu of Formal Methods*. Ed. by Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, and Einar Broch Johnsen. Lecture Notes in Computer Science 13360. Cham: Springer International Publishing, 2022, pp. 38–53. ISBN: 978-3-031-08166-8. DOI: [10.1007/978-3-031-08166-8_3](https://doi.org/10.1007/978-3-031-08166-8_3) (cit. on p. 36).
- [Buz+14] Igor Buzhinsky, Daniil Chivilikhin, Vladimir Ulyantsev, and Fedor Tsarev. “Improving the Quality of Supervised Finite-State Machine Construction Using Real-Valued Variables.” In: *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: Association for Computing Machinery, July 2014, pp. 1037–1040. ISBN: 978-1-4503-2881-4. DOI: [10.1145/2598394.2605679](https://doi.org/10.1145/2598394.2605679) (cit. on p. 33).
- [BV24] Robert Baumgartner and Sicco Verwer. “PDFa Distillation with Error Bound Guarantees.” In: *Implementation and Application of Automata*. Ed. by Szilárd Zsolt Fazekas. Lecture Notes in Computer Science 15015. Cham: Springer Nature Switzerland, 2024, pp. 51–65. ISBN: 978-3-031-71112-1. DOI: [10.1007/978-3-031-71112-1_4](https://doi.org/10.1007/978-3-031-71112-1_4) (cit. on p. 38).
- [Cam24] Robert C. Camp. *Benchmarking. The Search for Industry Best Practices that Lead to Superior Performance*. New York: Productivity Press, Nov. 2024. ISBN: 978-1-003-57887-1. DOI: [10.4324/9781003578871](https://doi.org/10.4324/9781003578871) (cit. on p. 3).
- [Car+22] Angela Carrera-Rivera, William Ochoa, Felix Larrinaga, and Ganix Lasa. “How-To Conduct a Systematic Literature Review: A Quick Guide for Computer Science Research.” In: *MethodsX* 9, 101895 (2022). ISSN: 2215-0161. DOI: [10.1016/j.mex.2022.101895](https://doi.org/10.1016/j.mex.2022.101895) (cit. on p. 76).
- [Cas+15] Sofia Cassel, Falk Howar, Bengt Jonsson, Maik Merten, and Bernhard Steffen. “A Succinct Canonical Register Automaton Model.” In: *Journal of Logical and Algebraic Methods in Programming* 84.1 (Jan. 2015), pp. 54–66. ISSN: 2352-2208. DOI: [10.1016/j.jlamp.2014.07.004](https://doi.org/10.1016/j.jlamp.2014.07.004) (cit. on pp. 1, 2, 7, 15, 17, 18, 26, 50).
- [Cas+16] Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. “Active Learning for Extended Finite State Machines.” In: *Formal Aspects of Computing* 28.2 (Apr. 2016), pp. 233–263. ISSN: 0934-5043. DOI: [10.1007/s00165-016-0355-5](https://doi.org/10.1007/s00165-016-0355-5) (cit. on pp. 44, 46, 55, 57).
- [CD12] Vincent Carnino and Sven De Felice. “Sampling Different Kinds of Acyclic Automata Using Markov Chains.” In: *Theoretical Computer Science* 450 (Sept. 2012), pp. 31–42. ISSN: 0304-3975. DOI: [10.1016/j.tcs.2012.04.025](https://doi.org/10.1016/j.tcs.2012.04.025) (cit. on p. 13).
- [CFJ15] Sofia Cassel, Howar Falk, and Bengt Jonsson. “RALib : A LearnLib Extension for Inferring EFMSs.” In: *DIFTS15: International Workshop on Design and Implementation of Formal Tools and Systems*. 2015 (cit. on p. 52).
- [CGV94] Antonio Castellanos, Isabel Galiano, and Enrique Vidal. “Application of OSTIA to Machine Translation Tasks.” In: *Grammatical Inference and Applications*. Ed. by Rafael C. Carrasco and Jose Oncina. Lecture Notes in Computer Science 862. Berlin, Heidelberg: Springer, 1994, pp. 93–105. ISBN: 978-3-540-48985-6. DOI: [10.1007/3-540-58473-0_140](https://doi.org/10.1007/3-540-58473-0_140) (cit. on p. 37).

- [Cha+04] Jean-Marc Champarnaud, Georges Hansel, Thomas Paranthoën, and Djelloul Ziadi. “Random Generation Models for NFAs.” In: *Journal of Automata, Languages and Combinatorics* 9.2–3 (2004), pp. 203–216. ISSN: 2567-3785. DOI: [10.25596/jalc-2004-203](https://doi.org/10.25596/jalc-2004-203) (cit. on p. 13).
- [Cha+08] Sagar Chaki, Edmund Clarke, Natasha Sharygina, and Nishant Sinha. “Verification of Evolving Software via Component Substitutability Analysis.” In: *Formal Methods in System Design* 32.3 (June 2008), pp. 235–266. ISSN: 1572-8102. DOI: [10.1007/s10703-008-0053-x](https://doi.org/10.1007/s10703-008-0053-x) (cit. on p. 34).
- [Cha+14] Georg Chalupar, Stefan Peherstorfer, Erik Poll, and Joeri de Ruiter. “Automated Reverse Engineering Using Lego®.” In: *8th USENIX Workshop on Offensive Technologies (WOOT14)*. San Diego, CA: USENIX Association, Aug. 2014. URL: <https://www.usenix.org/conference/woot14/workshop-program/presentation/chalupar> (cit. on p. 44).
- [CK02] Orlando Cicchello and Stefan C. Kremer. “Beyond EDSM.” In: *Grammatical Inference: Algorithms and Applications*. Ed. by Pieter Adriaans, Henning Fernau, and Menno van Zaanen. Lecture Notes in Computer Science 2484. Berlin, Heidelberg: Springer, 2002, pp. 37–48. ISBN: 978-3-540-45790-9. DOI: [10.1007/3-540-45790-9_4](https://doi.org/10.1007/3-540-45790-9_4) (cit. on p. 39).
- [CK98] Edward Y. C. Cheng and Michael Kaminski. “Context-Free Languages Over Infinite Alphabets.” In: *Acta Informatica* 35.3 (Mar. 1998), pp. 245–267. ISSN: 1432-0525. DOI: [10.1007/s002360050120](https://doi.org/10.1007/s002360050120) (cit. on p. 2).
- [CN12a] Arnaud Carayol and Cyril Nicaud. “Distribution of the Number of Accessible States in a Random Deterministic Automaton.” In: *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*. Ed. by Christoph Dürr and Thomas Wilke. Leibniz International Proceedings in Informatics (LIPIcs) 14. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012, pp. 194–205. ISBN: 978-3-939897-35-4. DOI: [10.4230/LIPIcs.STACS.2012.194](https://doi.org/10.4230/LIPIcs.STACS.2012.194). (Visited on 06/27/2025) (cit. on p. 13).
- [CN12b] Yingke Chen and Thomas Dyhre Nielsen. “Active Learning of Markov Decision Processes for System Verification.” In: *2012 11th International Conference on Machine Learning and Applications*. Ed. by M. Arif Wani, Taghi Khoshgoftaar, Xingquan (Hill) Zhu, and Naeem Seliya. Vol. 2. Institute of Electrical and Electronics Engineers, Dec. 2012, pp. 289–294. ISBN: 978-1-4673-4651-1. DOI: [10.1109/ICMLA.2012.158](https://doi.org/10.1109/ICMLA.2012.158) (cit. on p. 36).
- [CO94] Rafael C. Carrasco and Jose Oncina. “Learning Stochastic Regular Grammars by Means of a State Merging Method.” In: *Grammatical Inference and Applications*. Ed. by Rafael C. Carrasco and Jose Oncina. Lecture Notes in Computer Science 862. Berlin, Heidelberg: Springer, 1994, pp. 139–152. ISBN: 978-3-540-48985-6. DOI: [10.1007/3-540-58473-0_144](https://doi.org/10.1007/3-540-58473-0_144) (cit. on p. 37).
- [Cod+25] Cayden Codel, Katalin Fazekas, Marijn J. H. Heule, and Markus Iser, eds. *Proceedings of SAT Competition 2025. Solver and Benchmark Descriptions*. Aug. 2025. DOI: [10.34726/10379](https://doi.org/10.34726/10379) (cit. on p. 14).
- [Com+09] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. “Prospex: Protocol Specification Extraction.” In: *2009 30th IEEE Symposium on Security and Privacy*. Institute of Electrical and Electronics Engineers, May 2009, pp. 110–125. ISBN: 978-0-7695-3633-0. DOI: [10.1109/SP.2009.14](https://doi.org/10.1109/SP.2009.14) (cit. on p. 33).

- [Con79] Dennis M. Conti. *Findings of the Standard Benchmark Library Study Group*. Special Publication 500-38. Gaithersburg, MD: National Institute of Standards and Technology, Jan. 1979. URL: <https://www.nist.gov/publications/findings-standard-benchmark-library-study-group> (cit. on p. 5).
- [Cor+22] L  na  g Cornanguer, Christine Largou  t, Laurence Roz  , and Alexandre Termier. “TAG: Learning Timed Automata from Logs.” In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.4 (June 2022), pp. 3949–3958. ISSN: 2374-3468. DOI: [10.1609/aaai.v36i4.20311](https://doi.org/10.1609/aaai.v36i4.20311) (cit. on pp. 43, 48).
- [Cor25] Core Specification Working Group. *Bluetooth Core Specification*. Version v6.1. Bluetooth SIG, Apr. 29, 2025. URL: <https://www.bluetooth.com/specifications/specs/core-specification-6-1/> (cit. on pp. 34, 44).
- [Cox64] D. R. Cox. “Some Applications of Exponential Ordered Scores.” In: *Journal of the Royal Statistical Society: Series B (Methodological)* 26.1 (Jan. 1964), pp. 103–110. ISSN: 0035-9246. DOI: [10.1111/j.2517-6161.1964.tb00544.x](https://doi.org/10.1111/j.2517-6161.1964.tb00544.x) (cit. on p. 71).
- [CP05] Jean-Marc Champarnaud and Thomas Parantho  n. “Random Generation of DFAs.” In: *Theoretical Computer Science* 330.2 (Feb. 2005), pp. 221–235. ISSN: 0304-3975. DOI: [10.1016/j.tcs.2004.03.072](https://doi.org/10.1016/j.tcs.2004.03.072) (cit. on pp. 6, 9, 14, 83, 87, 100).
- [CP16] Christian Collberg and Todd A. Proebsting. “Repeatability in Computer Systems Research.” In: *Communications of the ACM* 59.3 (Feb. 2016), pp. 62–69. ISSN: 0001-0782. DOI: [10.1145/2812803](https://doi.org/10.1145/2812803) (cit. on p. 14).
- [CPS21] Thomas Colcombet, Daniela Petri   an, and Riccardo Stabile. “Learning Automata and Transducers: A Categorical Approach.” In: *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*. Ed. by Christel Baier and Jean Goubault-Larrecq. Leibniz International Proceedings in Informatics (LIPIcs) 183. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum f  r Informatik, 2021, 15:1–15:17. ISBN: 978-3-95977-175-7. DOI: [10.4230/LIPIcs.CSL.2021.15](https://doi.org/10.4230/LIPIcs.CSL.2021.15) (cit. on p. 32).
- [CU13] Daniil Chivilikhin and Vladimir Ulyantsev. “MuACOsM: A New Mutation-Based Ant Colony Optimization Algorithm for Learning Finite-State Machines.” In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. Ed. by Christian Blum. New York, NY, USA: Association for Computing Machinery, July 2013, pp. 511–518. ISBN: 978-1-4503-1963-8. DOI: [10.1145/2463372.2463440](https://doi.org/10.1145/2463372.2463440) (cit. on p. 37).
- [CUS14] Daniil Chivilikhin, Vladimir Ulyantsev, and Anatoly Shalyto. “Combining Exact and Metaheuristic Techniques for Learning Extended Finite-State Machines from Test Scenarios and Temporal Properties.” In: *2014 13th International Conference on Machine Learning and Applications*. Ed. by Xue-wen Chen, Guangzhi Qu, Plamen Angelov, Cesar Ferri, Jian-huang Lai, and M. Arif Wani. Institute of Electrical and Electronics Engineers, Dec. 2014, pp. 350–355. ISBN: 978-1-4799-7415-3. DOI: [10.1109/ICMLA.2014.62](https://doi.org/10.1109/ICMLA.2014.62) (cit. on p. 48).
- [CW98] Jonathan E. Cook and Alexander L. Wolf. “Discovering Models of Software Processes from Event-Based Data.” In: *ACM Transactions on Software Engineering Methodology* 7.3 (July 1998), pp. 215–249. ISSN: 1049-331X. DOI: [10.1145/287000.287001](https://doi.org/10.1145/287000.287001) (cit. on pp. 33, 37).

- [DAH25] Gabriel Dengler, Sven Apel, and Holger Hermanns. *Automata Learning – Expect Delays!* arXiv Preprint arXiv:2508.16384. Aug. 2025. DOI: [10.48550/arXiv.2508.16384](https://doi.org/10.48550/arXiv.2508.16384) (cit. on pp. 44, 47).
- [Dam+05] C. Damas, B. Lambeau, P. Dupont, and A. van Lamsweerde. “Generating Annotated Behavior Models from End-User Scenarios.” In: *IEEE Transactions on Software Engineering* 31.12 (Dec. 2005), pp. 1056–1073. ISSN: 1939-3520. DOI: [10.1109/TSE.2005.138](https://doi.org/10.1109/TSE.2005.138) (cit. on p. 38).
- [DAn24] Loris D’Antoni. *AutomatArk (Automata Benchmark)*. Oct. 4, 2024. URL: <https://github.com/lorisdanto/automatark> (cit. on p. 14).
- [Dem+21] Theo Demessance, Chongke Bi, Sonia Djebali, and Guillaume Guérard. “Hidden Markov Model to Predict Tourists Visited Places.” In: *2021 22nd IEEE International Conference on Mobile Data Management (MDM)*. Institute of Electrical and Electronics Engineers, June 2021, pp. 209–216. ISBN: 978-1-6654-2845-3. DOI: [10.1109/MDM52706.2021.00041](https://doi.org/10.1109/MDM52706.2021.00041) (cit. on p. 71).
- [DG20] Omar al Duhaiby and Jan Friso Groote. “Active Learning of Decomposable Systems.” In: *Proceedings of the 8th International Conference on Formal Methods in Software Engineering*. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 1–10. ISBN: 978-1-4503-7071-4. DOI: [10.1145/3372020.3391560](https://doi.org/10.1145/3372020.3391560) (cit. on p. 34).
- [DH21] Simon Dierl and Falk Howar. “A Taxonomy and Reductions for Common Register Automata Formalisms.” In: *Model Checking, Synthesis, and Learning*. Ed. by Ernst-Rüdiger Olderog, Bernhard Steffen, and Wang Yi. Lecture Notes in Computer Science 13030. Cham: Springer International Publishing, 2021, pp. 186–218. ISBN: 978-3-030-91384-7. DOI: [10.1007/978-3-030-91384-7_10](https://doi.org/10.1007/978-3-030-91384-7_10) (cit. on pp. v, 8, 15, 18, 19, 21, 23, 25–27).
- [DH22a] Simon Dierl and Falk Howar. “REACH on Register Automata via History Independence.” In: *Tests and Proofs*. Ed. by Laura Kovács and Karl Meinke. Lecture Notes in Computer Science 13361. Cham: Springer International Publishing, 2022, pp. 11–30. ISBN: 978-3-031-09827-7. DOI: [10.1007/978-3-031-09827-7_2](https://doi.org/10.1007/978-3-031-09827-7_2) (cit. on pp. v, 8, 9, 11, 49, 51, 79).
- [DH22b] Simon Dierl and Falk Howar. *Reach on Register Automata via History Independence – Replication Artifact*. Mar. 18, 2022. DOI: [10.5281/zenodo.6367981](https://doi.org/10.5281/zenodo.6367981) (cit. on p. 95).
- [DH25] Simon Dierl and Falk Howar. *Towards a Specification for Recombinable Benchmarks and Software Tools*. ing.grid Preprint. Universitäts- und Landesbibliothek Darmstadt, July 14, 2025. URL: <https://preprints.inggrid.org/repository/view/55/>. Submitted (cit. on pp. vi, 9, 89, 91, 92).
- [Die+21] Simon Dierl, Falk Howar, Malte Mues, Stefan Naujokat, and Till Schallau. “Do Away with the Frankensteinian Programs! A Proposal for a Genuine SE Education.” In: *2021 Third International Workshop on Software Engineering Education for the Next Generation (SEENG)*. Institute of Electrical and Electronics Engineers, May 2021, pp. 26–30. ISBN: 978-1-6654-3196-5. DOI: [10.1109/SEENG53126.2021.00012](https://doi.org/10.1109/SEENG53126.2021.00012) (cit. on p. vii).
- [Die+23a] 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: *NASA Formal Methods*. Ed. by Kristin Yvonne Rozier and Swarat Chaudhuri. Lecture Notes in Computer Science 13903. Cham: Springer Nature

- Switzerland, 2023, pp. 104–121. ISBN: 978-3-031-33170-1. DOI: [10.1007/978-3-031-33170-1_7](https://doi.org/10.1007/978-3-031-33170-1_7) (cit. on pp. v, 8, 11, 60, 63, 64).
- [Die+23b] 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 – Data and Replication Package*. Mar. 24, 2023. DOI: [10.5281/zenodo.7766789](https://doi.org/10.5281/zenodo.7766789) (cit. on p. 95).
- [Die+24] Simon Dierl, Paul Fiterau-Brostean, Falk Howar, Bengt Jonsson, Konstantinos Sagonas, and Fredrik Tåquist. “Scalable Tree-Based Register Automata Learning.” In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Bernd Finkbeiner and Laura Kovács. Lecture Notes in Computer Science 14571. Cham: Springer Nature Switzerland, 2024, pp. 87–108. ISBN: 978-3-031-57249-4. DOI: [10.1007/978-3-031-57249-4_5](https://doi.org/10.1007/978-3-031-57249-4_5) (cit. on pp. v, 8, 9, 11, 55–57, 83).
- [Die+26a] Simon Dierl, Paul Fiterău-Broștean, Falk Howar, Bengt Jonsson, Konstantinos Sagonas, and Fredrik Tåquist. “ SL^λ : A Scalable Algorithm for Register Automata Learning.” In: *Journal of Automated Reasoning* 70, 14 (Aug. 2026), p. 14. ISSN: 1573-0670. DOI: [10.1007/s10817-026-09758-9](https://doi.org/10.1007/s10817-026-09758-9) (cit. on pp. vi, 8, 9, 11, 55–57, 83).
- [Die+26b] Simon Dierl, Nastaran Kianersi, Falk Howar, and Sean Kauffman. “Learning Probabilistic Automata from Single Continuous-Valued System Logs.” In: *NASA Formal Methods*. Ed. by Jyotirmoy Deshmukh, Klaus Havelund, and Alessandro Pinto. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2026, pp. 170–191. ISBN: 978-3-032-28079-4. DOI: [10.1007/978-3-032-28079-4_8](https://doi.org/10.1007/978-3-032-28079-4_8) (cit. on pp. vi, 8, 11, 69, 70).
- [Dir+21] Directorate-General for Research and Innovation (European Commission), EOSC Executive Board, Oscar Corcho, Magnus Eriksson, Krzysztof Kurowski, Milan Ojsteršek, Christine Choirat, Mark van de Sanden, and Frederik Coppens. *EOSC Interoperability Framework: Report from the EOSC Executive Board Working Groups FAIR and Architecture*. Publications Office of the European Union, 2021. ISBN: 978-92-76-28949-4. DOI: [10.2777/620649](https://doi.org/10.2777/620649) (cit. on p. 14).
- [DL09] Stéphane Demri and Ranko Lazić. “LTL with the Freeze Quantifier and Register Automata.” In: *ACM Transactions on Computational Logic* 10.3, 16 (Apr. 2009). ISSN: 1529-3785. DOI: [10.1145/1507244.1507246](https://doi.org/10.1145/1507244.1507246) (cit. on pp. 12, 15, 26, 43, 45, 51).
- [DLT04] François Denis, Aurélien Lemay, and Alain Terlutte. “Learning Regular Languages Using RFSAs.” In: *Theoretical Computer Science* 313.2 (Feb. 2004), pp. 267–294. ISSN: 0304-3975. DOI: [10.1016/j.tcs.2003.11.008](https://doi.org/10.1016/j.tcs.2003.11.008) (cit. on pp. 39, 41).
- [DM93] Sreerupa Das and Michael C. Mozer. “A Unified Gradient-Descent/clustering Architecture for Finite State Machine Induction.” In: *Advances in Neural Information Processing Systems 6 (NIPS 1993)*. Ed. by J. Cowan, G. Tesauro, and J. Alspector. Morgan Kaufmann, 1993. ISBN: 1-55860-322-0. URL: <https://papers.nips.cc/paper/1993/hash/84f7e69969dea92a925508f7c1f9579a-Abstract.html> (cit. on p. 35).
- [DMS19] Carlos Diego N. Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simao. “Learning to Reuse: Adaptive Model Learning for Evolving Systems.” In: *Integrated Formal Methods*. Ed. by Wolfgang Ahrendt and Silvia Lizeth Tapia Tarifa. Lecture Notes in Computer Science 11918. Cham: Springer International Publishing, 2019, pp. 138–156. ISBN: 978-3-030-34968-4. DOI: [10.1007/978-3-030-34968-4_8](https://doi.org/10.1007/978-3-030-34968-4_8) (cit. on p. 34).

- [DPR18] Lesly-Ann Daniel, Erik Poll, and Joeri de Ruiter. “Inferring OpenVPN State Machines Using Protocol State Fuzzing.” In: *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. Institute of Electrical and Electronics Engineers, Apr. 2018, pp. 11–19. ISBN: 978-1-5386-5445-3. DOI: [10.1109/EuroSPW.2018.00009](https://doi.org/10.1109/EuroSPW.2018.00009) (cit. on p. 44).
- [Dup+08] Pierre Dupont, Bernard Lambeau, Christophe Damas, and Axel van Lamsweerde. “The QSM Algorithm and Its Application to Software Behavior Model Induction.” In: *Applied Artificial Intelligence* 22.1 (Feb. 2008), pp. 77–115. ISSN: 0883-9514. DOI: [10.1080/08839510701853200](https://doi.org/10.1080/08839510701853200) (cit. on pp. 36, 38, 40).
- [Edd22] Wesley Eddy. *Transmission Control Protocol (TCP)*. Internet Standard 7. RFC Editor, Aug. 2022. DOI: [10.17487/RFC9293](https://doi.org/10.17487/RFC9293) (cit. on pp. 33, 34, 43–45).
- [EM18] S. S. Emam and J. Miller. “Inferring Extended Probabilistic Finite-State Automaton Models from Software Executions.” In: *ACM Transactions on Software Engineering and Methodology* 27.1 (June 2018), 4:1–4:39. ISSN: 1049-331X. DOI: [10.1145/3196883](https://doi.org/10.1145/3196883) (cit. on p. 43).
- [ER60] Paul Erdős and Alfréd Rényi. “On the Evolution of Random Graphs.” In: *A Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei* 5 (1960), pp. 17–61. URL: <https://real-j.mtak.hu/511/> (cit. on p. 6).
- [FB96] Ned Freed and Nathaniel S. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. Request for Comments 2046. RFC Editor, Nov. 1996. DOI: [10.17487/RFC2046](https://doi.org/10.17487/RFC2046) (cit. on p. 91).
- [Fel+90] Jerome A. Feldman, George Lakoff, Andreas Stolcke, and Susan Hollbach Weber. “Miniature Language Acquisition: A Touchstone for Cognitive Science.” In: *Program of the Twelfth Annual Conference of the Cognitive Science Society, 25-28 July 1990, Cambridge, Massachusetts*. Ed. by John R. Anderson. New York: Psychology Press, 1990, pp. 686–693. URL: <https://www.taylorfrancis.com/chapters/10.4324/9781315807249-22> (cit. on p. 37).
- [FHS22] Tiago Ferreira, Gerco van Heerdt, and Alexandra Silva. “Tree-Based Adaptive Model Learning.” In: *A Journey from Process Algebra via Timed Automata to Model Learning*. Ed. by Nils Jansen, Mariëlle Stoelinga, and Petra van den Bos. Lecture Notes in Computer Science 13560. Cham: Springer Nature Switzerland, 2022, pp. 164–179. ISBN: 978-3-031-15629-8. DOI: [10.1007/978-3-031-15629-8_10](https://doi.org/10.1007/978-3-031-15629-8_10) (cit. on p. 40).
- [FHS25] Markus Frohme, Falk Howar, and Bernhard Steffen. “LearnLib: 10 Years Later.” In: *Computer Aided Verification*. Ed. by Ruzica Piskac and Zvonimir Rakamarić. Lecture Notes in Computer Science 15934. Cham: Springer Nature Switzerland, 2025, pp. 141–160. ISBN: 978-3-031-98685-7. DOI: [10.1007/978-3-031-98685-7_7](https://doi.org/10.1007/978-3-031-98685-7_7) (cit. on p. 14).
- [Fis+21] Iztok Fister, Janez Brest, Andres Iglesias, Akemi Galvez, Suash Deb, and Iztok Fister. “On Selection of a Benchmark by Determining the Algorithms’ Qualities.” In: *IEEE Access* 9 (2021), pp. 51166–51178. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2021.3058285](https://doi.org/10.1109/ACCESS.2021.3058285) (cit. on p. 12).
- [Fit+17] Paul Fiterău-Broștean, Toon Lenaerts, Erik Poll, Joeri de Ruiter, Frits Vaandrager, and Patrick Verleg. “Model Learning and Model Checking of SSH Implementations.” In: *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*. Ed. by Hakan Erdogmus and Klaus Havelund. New York, NY, USA: Association

- for Computing Machinery, July 2017, pp. 142–151. ISBN: 978-1-4503-5077-8. DOI: [10.1145/3092282.3092289](https://doi.org/10.1145/3092282.3092289) (cit. on p. 44).
- [Fit+20] Paul Fiterau-Broştean, Bengt Jonsson, Robert Merget, Joeri de Ruiter, Konstantinos Sagonas, and Juraj Somorovsky. “Analysis of DTLS Implementations Using Protocol State Fuzzing.” In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2523–2540. ISBN: 978-1-939133-17-5. URL: <https://www.usenix.org/conference/usenixsecurity20/presentation/fiterau-brostean> (cit. on p. 44).
- [Fit+22] Paul Fiterău-Broştean, Bengt Jonsson, Konstantinos Sagonas, and Fredrik Tåquist. “DTLS-Fuzzer: A DTLS Protocol State Fuzzer.” In: *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. Institute of Electrical and Electronics Engineers, Apr. 2022, pp. 456–458. ISBN: 978-1-6654-6679-0. DOI: [10.1109/ICST53961.2022.00051](https://doi.org/10.1109/ICST53961.2022.00051) (cit. on p. 57).
- [FJV14] Paul Fiterău-Broştean, Ramon Janssen, and Frits Vaandrager. “Learning Fragments of the TCP Network Protocol.” In: *Formal Methods for Industrial Critical Systems*. Ed. by Frédéric Lang and Francesco Flammini. Lecture Notes in Computer Science 8718. Cham: Springer International Publishing, 2014, pp. 78–93. ISBN: 978-3-319-10702-8. DOI: [10.1007/978-3-319-10702-8_6](https://doi.org/10.1007/978-3-319-10702-8_6) (cit. on p. 44).
- [FJV16] Paul Fiterău-Broştean, Ramon Janssen, and Frits Vaandrager. “Combining Model Learning and Model Checking to Analyze TCP Implementations.” In: *Computer Aided Verification*. Ed. by Swarat Chaudhuri and Azadeh Farzan. Lecture Notes in Computer Science 9780. Cham: Springer International Publishing, 2016, pp. 454–471. ISBN: 978-3-319-41540-6. DOI: [10.1007/978-3-319-41540-6_25](https://doi.org/10.1007/978-3-319-41540-6_25) (cit. on p. 44).
- [FMR18] Miguel Ferreira, Nelma Moreira, and Rogério Reis. “Forward Injective Finite Automata: Exact and Random Generation of Nonisomorphic NFAs.” In: *Descriptive Complexity of Formal Systems*. Ed. by Stavros Konstantinidis and Giovanni Pighizzini. Lecture Notes in Computer Science 10952. Cham: Springer International Publishing, 2018, pp. 88–100. ISBN: 978-3-319-94631-3. DOI: [10.1007/978-3-319-94631-3_8](https://doi.org/10.1007/978-3-319-94631-3_8) (cit. on p. 14).
- [Fra+25] Florian Frank, Stefan Milius, Jurriaan Rot, and Henning Urbat. *Learning Automata with Name Allocation*. arXiv Preprint arXiv:2502.11947. arXiv, Feb. 2025. DOI: [10.48550/arXiv.2502.11947](https://doi.org/10.48550/arXiv.2502.11947) (cit. on p. 43).
- [Fro19] Markus Theo Frohme. *Active Automata Learning with Adaptive Distinguishing Sequences*. arXiv Preprint arXiv:1902.01139. arXiv, Feb. 2019. DOI: [10.48550/arXiv.1902.01139](https://doi.org/10.48550/arXiv.1902.01139) (cit. on pp. 34, 40).
- [FS21] Markus Frohme and Bernhard Steffen. “Compositional Learning of Mutually Recursive Procedural Systems.” In: *International Journal on Software Tools for Technology Transfer* 23.4 (Aug. 2021), pp. 521–543. ISSN: 1433-2787. DOI: [10.1007/s10009-021-00634-y](https://doi.org/10.1007/s10009-021-00634-y) (cit. on p. 36).
- [Gar+13] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. “CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes.” In: *International Journal on Software Tools for Technology Transfer* 15.2 (Apr. 2013), pp. 89–107. ISSN: 1433-2787. DOI: [10.1007/s10009-012-0244-z](https://doi.org/10.1007/s10009-012-0244-z) (cit. on pp. 38, 47).

- [Gar+14] S. García, M. Grill, J. Stiborek, and A. Zunino. “An Empirical Comparison of Botnet Detection Methods.” In: *Computers & Security* 45 (Sept. 2014), pp. 100–123. ISSN: 0167-4048. DOI: [10.1016/j.cose.2014.05.011](https://doi.org/10.1016/j.cose.2014.05.011) (cit. on p. 38).
- [Gar+20] Bharat Garhewal, Frits Vaandrager, Falk Howar, Timo Schrijvers, Toon Lenaerts, and Rob Smits. “Grey-Box Learning of Register Automata.” In: *Integrated Formal Methods*. Ed. by Brijesh Dongol and Elena Troubitsyna. Lecture Notes in Computer Science 12546. Cham: Springer International Publishing, 2020, pp. 22–40. ISBN: 978-3-030-63461-2. DOI: [10.1007/978-3-030-63461-2_2](https://doi.org/10.1007/978-3-030-63461-2_2) (cit. on pp. 44, 46, 47).
- [Gay24] Alicia Gayda. “Processing Counterexamples in Active Learning of Register Automata.” MA thesis. TU Dortmund University, Apr. 2024 (cit. on p. 56).
- [GDS19] Ulrich Thomas Gabor, Simon Dierl, and Olaf Spinczyk. “Spectrum-Based Fault Localization in Deployed Embedded Systems with Driver Interaction Models.” In: *Computer Safety, Reliability, and Security*. Ed. by Alexander Romanovsky, Elena Troubitsyna, and Friedemann Bitsch. Lecture Notes in Computer Science 11698. Cham: Springer International Publishing, 2019, pp. 97–112. ISBN: 978-3-030-26601-1. DOI: [10.1007/978-3-030-26601-1_7](https://doi.org/10.1007/978-3-030-26601-1_7) (cit. on pp. vii, 33).
- [GF07] Sezer Gören and F. Joel Ferguson. “On State Reduction of Incompletely Specified Finite State Machines.” In: *Computers & Electrical Engineering* 33.1 (Jan. 2007), pp. 58–69. ISSN: 0045-7906. DOI: [10.1016/j.compeleceng.2006.06.001](https://doi.org/10.1016/j.compeleceng.2006.06.001) (cit. on p. 37).
- [GJL10] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. “Learning of Event-Recording Automata.” In: *Theoretical Computer Science* 411.47 (Oct. 2010), pp. 4029–4054. ISSN: 0304-3975. DOI: [10.1016/j.tcs.2010.07.008](https://doi.org/10.1016/j.tcs.2010.07.008) (cit. on p. 43).
- [GJP06] Olga Grinchtein, Bengt Jonsson, and Paul Pettersson. “Inference of Event-Recording Automata Using Timed Decision Trees.” In: *CONCUR 2006 – Concurrency Theory*. Ed. by Christel Baier and Holger Hermanns. Lecture Notes in Computer Science 4137. Berlin, Heidelberg: Springer, 2006, pp. 435–449. ISBN: 978-3-540-37377-3. DOI: [10.1007/11817949_29](https://doi.org/10.1007/11817949_29) (cit. on p. 43).
- [GL65] A. Grasselli and F. Luccio. “A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks.” In: *IEEE Transactions on Electronic Computers* EC-14.3 (June 1965), pp. 350–359. ISSN: 0367-7508. DOI: [10.1109/PGEC.1965.264140](https://doi.org/10.1109/PGEC.1965.264140) (cit. on p. 32).
- [Gol67] E. Mark Gold. “Language Identification in the Limit.” In: *Information and Control* 10.5 (May 1967), pp. 447–474. ISSN: 0019-9958. DOI: [10.1016/S0019-9958\(67\)91165-5](https://doi.org/10.1016/S0019-9958(67)91165-5) (cit. on p. 1).
- [Gol72] E. Mark Gold. “System Identification via State Characterization.” In: *Automatica* 8.5 (Sept. 1972), pp. 621–636. ISSN: 0005-1098. DOI: [10.1016/0005-1098\(72\)90033-7](https://doi.org/10.1016/0005-1098(72)90033-7) (cit. on p. 32).
- [GPY02] Alex Groce, Doron Peled, and Mihalis Yannakakis. “Adaptive Model Checking.” In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Joost-Pieter Katoen and Perdita Stevens. Lecture Notes in Computer Science 2280. Berlin, Heidelberg: Springer, 2002, pp. 357–370. ISBN: 978-3-540-46002-2. DOI: [10.1007/3-540-46002-0_25](https://doi.org/10.1007/3-540-46002-0_25) (cit. on p. 34).

- [Gro+18] Roland Groz, Adenilso Simao, Alexandre Petrenko, and Catherine Oriat. “Inferring FSM Models of Systems without Reset.” In: *Machine Learning for Dynamic Software Analysis. Potentials and Limits*. Ed. by Amel Bennaceur, Reiner Hähnle, and Karl Meinke. Lecture Notes in Computer Science 11026. Cham: Springer International Publishing, 2018, pp. 178–201. ISBN: 978-3-319-96562-8. DOI: [10.1007/978-3-319-96562-8_7](https://doi.org/10.1007/978-3-319-96562-8_7) (cit. on p. 40).
- [Gün+12] Canan Güniçen, Uraz Cengiz Türker, Hasan Ural, and Hüsnü Yenigün. “Generating Preset Distinguishing Sequences Using SAT.” In: *Computer and Information Sciences II*. Ed. by Erol Gelenbe, Ricardo Lent, and Georgia Sakellari. London: Springer, 2012, pp. 487–493. ISBN: 978-1-4471-2155-8. DOI: [10.1007/978-1-4471-2155-8_62](https://doi.org/10.1007/978-1-4471-2155-8_62) (cit. on p. 40).
- [Ha+11] Eun Ha, Jonathan Rowe, Bradford Mott, and James Lester. “Goal Recognition with Markov Logic Networks for Player-Adaptive Games.” In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 7.1* (Oct. 9, 2011), pp. 32–39. ISSN: 2334-0924. DOI: [10.1609/aiide.v7i1.12434](https://doi.org/10.1609/aiide.v7i1.12434) (cit. on p. 37).
- [Hac+91] G. D. Hachtel, J.-K. Rho, F. Somenzi, and R. Jacoby. “Exact and Heuristic Algorithms for the Minimization of Incompletely Specified State Machines.” In: *Proceedings of the European Conference on Design Automation*. Institute of Electrical and Electronics Engineers, Feb. 1991, pp. 184–191. DOI: [10.1109/EDAC.1991.206387](https://doi.org/10.1109/EDAC.1991.206387) (cit. on p. 37).
- [Har26] Moritz Hardt. *The Emerging Science of Machine Learning Benchmarks*. Princeton University Press, Oct. 6, 2026. ISBN: 978-0-691-28429-3 (cit. on p. 12).
- [HC15] Robert M. Hierons and Uraz Cengiz Türker. “Incomplete Distinguishing Sequences for Finite State Machines.” In: *The Computer Journal* 58.11 (Nov. 2015), pp. 3089–3113. ISSN: 0010-4620. DOI: [10.1093/comjnl/bxv041](https://doi.org/10.1093/comjnl/bxv041) (cit. on pp. 38, 40).
- [HDE06] Amaury Habrard, François Denis, and Yann Esposito. “Using Pseudo-Stochastic Rational Languages in Probabilistic Grammatical Inference.” In: *Grammatical Inference: Algorithms and Applications*. Ed. by Yasubumi Sakakibara, Satoshi Kobayashi, Kengo Sato, Tetsuro Nishino, and Etsuji Tomita. Lecture Notes in Computer Science 4201. Berlin, Heidelberg: Springer, 2006, pp. 112–124. ISBN: 978-3-540-45265-2. DOI: [10.1007/11872436_10](https://doi.org/10.1007/11872436_10) (cit. on p. 35).
- [Hen+25] Leo Henry, Thomas Neele, Mohammad Mousavi, and Matteo Sammartino. *Compositional Active Learning of Synchronous Systems through Automated Alphabet Refinement*. arXiv Preprint arXiv:2504.16624. arXiv, Apr. 2025. DOI: [10.48550/arXiv.2504.16624](https://doi.org/10.48550/arXiv.2504.16624) (cit. on pp. 38, 40).
- [Hie+09] Robert M. Hierons, Guy-Vincent Jourdan, Hasan Ural, and Husnu Yenigun. “Checking Sequence Construction Using Adaptive and Preset Distinguishing Sequences.” In: *2009 Seventh IEEE International Conference on Software Engineering and Formal Methods*. Ed. by Dang Van Hung and Padmanabhan Krishnan. Institute of Electrical and Electronics Engineers, Nov. 2009, pp. 157–166. ISBN: 978-0-7695-3870-9. DOI: [10.1109/SEFM.2009.12](https://doi.org/10.1109/SEFM.2009.12) (cit. on p. 40).
- [Hig05] Colin de la Higuera. “A Bibliographical Study of Grammatical Inference.” In: *Pattern Recognition* 38.9 (Sept. 2005), pp. 1332–1348. ISSN: 0031-3203. DOI: [10.1016/j.patcog.2005.01.003](https://doi.org/10.1016/j.patcog.2005.01.003) (cit. on p. 13).

- [Hig10] Colin de la Higuera. *Grammatical Inference. Learning Automata and Grammars*. Cambridge University Press, Apr. 2010. ISBN: 978-0-521-76316-5. DOI: [10.1017/CBO9781139194655](https://doi.org/10.1017/CBO9781139194655) (cit. on p. 13).
- [Hir+93] L. Hirschman, M. Bates, D. Dahl, W. Fisher, J. Garofolo, D. Pallett, K. Hunicke-Smith, P. Price, A. Rudnick, and E. Tzoukermann. “Multi-Site Data Collection and Evaluation in Spoken Language Understanding.” In: *Human Language Technology. Proceedings of a Workshop Held at Plainsboro, New Jersey, March 21-24, 1993*. San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1993. ISBN: 978-1-55860-324-0. URL: <https://aclanthology.org/H93-1004/> (cit. on p. 37).
- [HJ15] Pierre-Cyrille Héam and Jean-Luc Joly. “Random Generation and Enumeration of Accessible Deterministic Real-Time Pushdown Automata.” In: *Implementation and Application of Automata*. Ed. by Frank Drewes. Lecture Notes in Computer Science 9223. Cham: Springer International Publishing, 2015, pp. 153–164. ISBN: 978-3-319-22360-5. DOI: [10.1007/978-3-319-22360-5_13](https://doi.org/10.1007/978-3-319-22360-5_13) (cit. on p. 13).
- [HJM20] Léo Henry, Thierry Jéron, and Nicolas Markey. “Active Learning of Timed Automata with Unobservable Resets.” In: *Formal Modeling and Analysis of Timed Systems*. Ed. by Nathalie Bertrand and Nils Jansen. Cham: Springer International Publishing, 2020, pp. 144–160. ISBN: 978-3-030-57628-8. DOI: [10.1007/978-3-030-57628-8_9](https://doi.org/10.1007/978-3-030-57628-8_9) (cit. on p. 43).
- [HM96] Hiroyuki Higuchi and Yusuke Matsunaga. “A Fast State Reduction Algorithm for Incompletely Specified Finite State Machines.” In: *33rd Design Automation Conference, 1996*. Institute of Electrical and Electronics Engineers, June 1996, pp. 463–466. ISBN: 0-7803-3294-6. DOI: [10.1109/DAC.1996.545620](https://doi.org/10.1109/DAC.1996.545620) (cit. on p. 37).
- [HMQ13] Thomas Hanneforth, Andreas Maletti, and Daniel Quernheim. “Random Generation of Nondeterministic Finite-State Tree Automata.” In: *Proceedings Second International Workshop on Trends in Tree Automata and Tree Transducers*. Ed. by Sebastian Maneth. Electronic Proceedings in Theoretical Computer Science 134. Nov. 2013, pp. 11–16. DOI: [10.4204/EPTCS.134.2](https://doi.org/10.4204/EPTCS.134.2) (cit. on p. 13).
- [HMS03] Hardi Hungar, Tiziana Margaria, and Bernhard Steffen. “Test-Based Model Generation for Legacy Systems.” In: *International Test Conference, 2003. ITC 2003*. Institute of Electrical and Electronics Engineers, Sept. 2003, pp. 150–159. ISBN: 0-7803-8106-8. DOI: [10.1109/TEST.2003.1271205](https://doi.org/10.1109/TEST.2003.1271205) (cit. on p. 34).
- [HNS10] Pierre-Cyrille Héam, Cyril Nicaud, and Sylvain Schmitz. “Parametric Random Generation of Deterministic Tree Automata.” In: *Theoretical Computer Science* 411.38 (Aug. 2010), pp. 3469–3480. ISSN: 0304-3975. DOI: [10.1016/j.tcs.2010.05.036](https://doi.org/10.1016/j.tcs.2010.05.036) (cit. on p. 13).
- [Hoe63] Wassily Hoeffding. “Probability Inequalities for Sums of Bounded Random Variables.” In: *Journal of the American Statistical Association* 58.301 (Mar. 1963), pp. 13–30. ISSN: 0162-1459. DOI: [10.1080/01621459.1963.10500830](https://doi.org/10.1080/01621459.1963.10500830) (cit. on p. 71).
- [Hop71] John Hopcroft. “An $n \log n$ Algorithm for Minimizing States in a Finite Automaton.” In: *Theory of Machines and Computations*. Ed. by Zvi Kohavi and Azaria Paz. Academic Press, Jan. 1971, pp. 189–196. ISBN: 978-0-12-417750-5. DOI: [10.1016/B978-0-12-417750-5.50022-1](https://doi.org/10.1016/B978-0-12-417750-5.50022-1). (Visited on 06/27/2025) (cit. on pp. 35, 65).

- [HOV96] Colin de la Higuera, José Oncina, and Enrique Vidal. “Identification of DFA: Data-Dependent Versus Data-Independent Algorithms.” In: *Grammatical Interference: Learning Syntax from Sentences*. Ed. by Laurent Miclet and Colin de la Higuera. Lecture Notes in Computer Science 1147. Berlin, Heidelberg: Springer, 1996, pp. 313–325. ISBN: 978-3-540-70678-6. DOI: [10.1007/BFb0033365](https://doi.org/10.1007/BFb0033365) (cit. on p. 35).
- [How+12a] Falk Howar, Malte Isberner, Bernhard Steffen, Oliver Bauer, and Bengt Jonsson. “Inferring Semantic Interfaces of Data Structures.” In: *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*. Ed. by Tiziana Margaria and Bernhard Steffen. Lecture Notes in Computer Science 7609. Berlin, Heidelberg: Springer, 2012, pp. 554–571. ISBN: 978-3-642-34026-0. DOI: [10.1007/978-3-642-34026-0_41](https://doi.org/10.1007/978-3-642-34026-0_41) (cit. on p. 46).
- [How+12b] Falk Howar, Maik Merten, Bernhard Steffen, and Tiziana Margaria. “Practical Aspects of Active Automata Learning.” In: *Formal Methods for Industrial Critical Systems*. Ed. by Stefania Gnesi and Tiziana Margaria. John Wiley & Sons, Ltd, 2012, pp. 235–267. ISBN: 978-1-118-45989-8. DOI: [10.1002/978-1-118-45989-8.ch11](https://doi.org/10.1002/978-1-118-45989-8.ch11) (cit. on p. 13).
- [How+21] Falk Howar, Marc Jasper, Malte Mues, David Schmidt, and Bernhard Steffen. “The RERS Challenge: Towards Controllable and Scalable Benchmark Synthesis.” In: *International Journal on Software Tools for Technology Transfer* 23.6 (Dec. 2021), pp. 917–930. ISSN: 1433-2787. DOI: [10.1007/s10009-021-00617-z](https://doi.org/10.1007/s10009-021-00617-z) (cit. on p. 7).
- [How12] Falk M. Howar. “Active Learning of Interface Programs.” PhD thesis. Technische Universität Dortmund, June 2012. DOI: [10.17877/DE290R-4817](https://doi.org/10.17877/DE290R-4817) (cit. on p. 38).
- [HS18] Falk Howar and Bernhard Steffen. “Active Automata Learning in Practice. An Annotated Bibliography of the Years 2011 to 2016.” In: *Machine Learning for Dynamic Software Analysis. Potentials and Limits*. Ed. by Amel Bennaceur, Reiner Hähnle, and Karl Meinke. Lecture Notes in Computer Science 11026. Cham: Springer International Publishing, 2018, pp. 123–148. ISBN: 978-3-319-96562-8. DOI: [10.1007/978-3-319-96562-8_5](https://doi.org/10.1007/978-3-319-96562-8_5) (cit. on p. 13).
- [HS22] Falk Howar and Bernhard Steffen. “Active Automata Learning As Black-Box Search And lazy Partition Refinement.” In: *A Journey from Process Algebra via Timed Automata to Model Learning*. Ed. by Nils Jansen, Mariëlle Stoelinga, and Petra van den Bos. Lecture Notes in Computer Science 13560. Cham: Springer Nature Switzerland, 2022, pp. 321–338. ISBN: 978-3-031-15629-8. DOI: [10.1007/978-3-031-15629-8_17](https://doi.org/10.1007/978-3-031-15629-8_17) (cit. on pp. 38, 55, 56).
- [HSS17] Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. “CALF: Categorical Automata Learning Framework.” In: *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*. Ed. by Valentin Goranko and Mads Dam. Leibniz International Proceedings in Informatics (LIPIcs) 82. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017, 29:1–29:24. ISBN: 978-3-95977-045-3. DOI: [10.4230/LIPIcs.CSL.2017.29](https://doi.org/10.4230/LIPIcs.CSL.2017.29) (cit. on pp. 30, 32).
- [HV10] Marijn J. H. Heule and Sicco Verwer. “Exact DFA Identification Using SAT Solvers.” In: *Grammatical Inference: Theoretical Results and Applications*. Ed. by José M. Sempere and Pedro García. Lecture Notes in Computer Science 6339. Berlin, Heidelberg: Springer, 2010, pp. 66–79. ISBN: 978-3-642-15488-1. DOI: [10.1007/978-3-642-15488-1_7](https://doi.org/10.1007/978-3-642-15488-1_7) (cit. on p. 37).

- [HV13] Marijn J. H. Heule and Sicco Verwer. “Software Model Synthesis Using Satisfiability Solvers.” In: *Empirical Software Engineering* 18.4 (Aug. 2013), pp. 825–856. ISSN: 1573-7616. DOI: [10.1007/s10664-012-9222-z](https://doi.org/10.1007/s10664-012-9222-z) (cit. on p. 37).
- [HXB04] Heng Hu, Hong-Xi Xue, and Ji-Nian Bian. “HSM2: A New Heuristic State Minimization Algorithm for Finite State Machine.” In: *Journal of Computer Science and Technology* 19.5 (Sept. 2004), pp. 729–733. ISSN: 1860-4749. DOI: [10.1007/BF02945600](https://doi.org/10.1007/BF02945600) (cit. on pp. 35, 37).
- [IGO12] Muhammad Naeem Irfan, Roland Groz, and Catherine Oriat. “Improving Model Inference of Black Box Components Having Large Input Test Set.” In: *Proceedings of the Eleventh International Conference on Grammatical Inference*. Ed. by Jeffrey Heinz, Colin Higuera, and Tim Oates. Proceedings of Machine Learning Research 21. PMLR, Sept. 2012, pp. 133–138. URL: <https://proceedings.mlr.press/v21/irfan12a.html> (cit. on pp. 36, 40).
- [IHS13] Malte Isberner, Falk Howar, and Bernhard Steffen. “Inferring Automata with State-Local Alphabet Abstractions.” In: *NASA Formal Methods*. Ed. by Guillaume Brat, Neha Rungta, and Arnaud Venet. Lecture Notes in Computer Science 7871. Berlin, Heidelberg: Springer, 2013, pp. 124–138. ISBN: 978-3-642-38088-4. DOI: [10.1007/978-3-642-38088-4_9](https://doi.org/10.1007/978-3-642-38088-4_9) (cit. on pp. 44, 46, 47).
- [IHS14] Malte Isberner, Falk Howar, and Bernhard Steffen. “The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning.” In: *Runtime Verification*. Ed. by Borzoo Bonakdarpour and Scott A. Smolka. Lecture Notes in Computer Science 8734. Cham: Springer International Publishing, 2014, pp. 307–322. ISBN: 978-3-319-11164-3. DOI: [10.1007/978-3-319-11164-3_26](https://doi.org/10.1007/978-3-319-11164-3_26) (cit. on pp. 38, 41, 65).
- [Iso+20] Jon Ison, Hervé Ménager, Bryan Brancotte, Erik Jaaniso, Ahto Salumets, Tomáš Raček, Anna-Lena Lamprecht, Magnus Palmblad, Matúš Kalaš, Piotr Chmura, John M. Hancock, Veit Schwämmle, and Hans-Ioan Ienasescu. “Community Curation of Bioinformatics Software and Data Resources.” In: *Briefings in Bioinformatics* 21.5 (Sept. 2020), pp. 1697–1705. ISSN: 1477-4054. DOI: [10.1093/bib/bbz075](https://doi.org/10.1093/bib/bbz075) (cit. on p. 14).
- [IT21] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Request for Comments 9000. Oct. 15, 2021. DOI: [10.17487/RFC9000](https://doi.org/10.17487/RFC9000) (cit. on p. 44).
- [Ito81] Stephen Y. Itoga. “A New Heuristic for Inferring Regular Grammars.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-3.2 (Mar. 1981), pp. 191–197. ISSN: 1939-3539. DOI: [10.1109/TPAMI.1981.4767078](https://doi.org/10.1109/TPAMI.1981.4767078) (cit. on p. 35).
- [Ive+00] Torsten K. Iversen, Kåre J. Kristoffersen, Kim G. Larsen, Morten Laursen, Rune G. Madsen, Steffen K. Mortensen, Paul Pettersson, and Chris B. Thomasen. “Model-Checking Real-Time Control Programs. Verifying LEGO® MINDSTORMS™ systems using UPPAAL.” In: *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000*. Institute of Electrical and Electronics Engineers, June 2000, pp. 147–155. ISBN: 978-0-7695-0734-7. DOI: [10.1109/EMRTS.2000.854002](https://doi.org/10.1109/EMRTS.2000.854002) (cit. on pp. 5, 63, 67).
- [Jac+90] Richard H. F. Jackson, Paul T. Boggs, Stephen G. Nash, and Susan Powell. “Guidelines for Reporting Results of Computational Experiments. Report of the Ad Hoc Committee.” In: *Mathematical Programming* 49.1 (Nov. 1990), pp. 413–425. ISSN: 1436-4646. DOI: [10.1007/BF01588801](https://doi.org/10.1007/BF01588801). URL: <https://doi.org/10.1007/BF01588801> (visited on 04/19/2026) (cit. on p. 11).

- [Jef+92] David Jefferson, Robert Collins, Claus Cooper, Michael Dyer, Margot Flowers, Richard Korf, Charles Taylor, and Alan Wang. “Evolution As a Theme in Artificial Life: The Genesys/Tracker System.” In: *Artificial Life II*. Ed. by Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen. Santa Fe Institute Studies in the Sciences of Complexity X. Addison-Wesley, 1992, pp. 511–548. ISBN: 978-0-89791-653-0 (cit. on p. 37).
- [Jep+20] Natasha Yogananda Jeppu, Thomas Melham, Daniel Kroening, and John O’Leary. “Learning Concise Models from Long Execution Traces.” In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. Institute of Electrical and Electronics Engineers, July 2020, pp. 1–6. ISBN: 978-1-7281-1085-1. DOI: [10.1109/DAC18072.2020.9218613](https://doi.org/10.1109/DAC18072.2020.9218613) (cit. on pp. 33, 35).
- [Jia+24] Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R. Lyu. “A Large-Scale Evaluation for Log Parsing Techniques: How Far Are We?” In: *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. Ed. by Maria Christakis and Michael Pradel. New York, NY, USA: Association for Computing Machinery, Sept. 2024, pp. 223–234. ISBN: 979-8-4007-0612-7. DOI: [10.1145/3650212.3652123](https://doi.org/10.1145/3650212.3652123) (cit. on p. 93).
- [Jos01] Mathai Joseph. “Time and Real-Time.” In: *Real-Time Systems Specification, Verification and Analysis*. Revised version with corrections. Tata Research Development & Design Centre, June 2001. Chap. 1, pp. 1–14. URL: <https://www.cs.york.ac.uk/rts/books/RTS-SVABook.html> (cit. on p. 38).
- [Jos65] E. O. Joslin. “Evaluation and Performance of Computers: Application Benchmarks: The Key to Meaningful Computer Evaluations.” In: *ACM ’65: Proceedings of the 1965 20th National Conference*. Ed. by Lewis Winner. New York, NY, USA: Association for Computing Machinery, 1965, pp. 27–37. ISBN: 978-1-4503-7495-8. DOI: [10.1145/800197.806031](https://doi.org/10.1145/800197.806031) (cit. on p. 3).
- [JS14] Bart Jacobs and Alexandra Silva. “Automata Learning: A Categorical Perspective.” In: *Horizons of the Mind. A Tribute to Prakash Panangaden*. Ed. by Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten. Lecture Notes in Computer Science 8464. Cham: Springer International Publishing, 2014, pp. 384–406. ISBN: 978-3-319-06880-0. DOI: [10.1007/978-3-319-06880-0_20](https://doi.org/10.1007/978-3-319-06880-0_20) (cit. on p. 32).
- [Kam+94] Timothy Kam, Tiziano Villa, Robert Brayton, and Alberto Sangiovanni-Vincentelli. “A Fully Implicit Algorithm for Exact State Minimization.” In: *Proceedings of the 31st Annual Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, June 1994, pp. 684–690. ISBN: 978-0-89791-653-0. DOI: [10.1145/196244.196615](https://doi.org/10.1145/196244.196615) (cit. on pp. 33, 35, 37, 39).
- [Kar16] Ahmet Kara. “Logics on Data Words.” PhD thesis. Universität Dortmund, 2016. DOI: [10.17877/DE290R-17260](https://doi.org/10.17877/DE290R-17260) (cit. on p. 13).
- [KD02] Christopher Kermorvant and Pierre Dupont. “Stochastic Grammatical Inference with Multinomial Tests.” In: *Grammatical Inference: Algorithms and Applications*. Ed. by Pieter Adriaans, Henning Fernau, and Menno van Zaanen. Lecture Notes in Computer Science 2484. Berlin, Heidelberg: Springer, 2002, pp. 149–160. ISBN: 978-3-540-45790-9. DOI: [10.1007/3-540-45790-9_12](https://doi.org/10.1007/3-540-45790-9_12) (cit. on p. 37).

- [KE95] James Kennedy and Russell Eberhart. “Particle Swarm Optimization.” In: *Proceedings of ICNN’95 - International Conference on Neural Networks*. Vol. 4. Institute of Electrical and Electronics Engineers, Nov. 1995, pp. 1942–1948. DOI: [10.1109/ICNN.1995.488968](https://doi.org/10.1109/ICNN.1995.488968) (cit. on p. 72).
- [Kel+90] M. I. Kellner, P. H. Feiler, A. Finkelstein, T. Katayama, L. J. Osterweil, M. H. Penedo, and H. D. Rombach. “Software Process Modeling Example Problem.” In: *Proceedings of the 6th International Software Process Workshop ‘Support for the Software Process’*. Institute of Electrical and Electronics Engineers, Oct. 1990, pp. 19–29. ISBN: 0-8186-2495-7. DOI: [10.1109/ISPW.1990.659568](https://doi.org/10.1109/ISPW.1990.659568) (cit. on p. 37).
- [Kel71] J. Kella. “Sequential Machine Identification.” In: *IEEE Transactions on Computers* C-20.3 (Mar. 1971), pp. 332–338. ISSN: 1557-9956. DOI: [10.1109/T-C.1971.223241](https://doi.org/10.1109/T-C.1971.223241) (cit. on p. 32).
- [KF94] Michael Kaminski and Nissim Francez. “Finite-Memory Automata.” In: *Theoretical Computer Science* 134.2 (Nov. 1994), pp. 329–363. ISSN: 0304-3975. DOI: [10.1016/0304-3975\(94\)90242-9](https://doi.org/10.1016/0304-3975(94)90242-9) (cit. on pp. 2, 7, 12, 15, 18, 26, 30).
- [KFE12] R. Killick, P. Fearnhead, and I. A. Eckley. “Optimal Detection of Changepoints With a Linear Computational Cost.” In: *Journal of the American Statistical Association* 107.500 (Dec. 2012), pp. 1590–1598. ISSN: 0162-1459. DOI: [10.1080/01621459.2012.737745](https://doi.org/10.1080/01621459.2012.737745) (cit. on p. 70).
- [KJR25] Loes Kruger, Sebastian Junges, and Jurriaan Rot. “State Matching and Multiple References in Adaptive Active Automata Learning.” In: *Formal Methods*. Ed. by André Platzer, Kristin Yvonne Rozier, Matteo Pradella, and Matteo Rossi. Lecture Notes in Computer Science 14933. Cham: Springer Nature Switzerland, 2025, pp. 267–284. ISBN: 978-3-031-71162-6. DOI: [10.1007/978-3-031-71162-6_14](https://doi.org/10.1007/978-3-031-71162-6_14) (cit. on pp. 34, 38).
- [KKG22] Paul Kogel, Verena Klös, and Sabine Glesner. “TTT/ik: Learning Accurate Mealy Automata Efficiently with an Imprecise Symbol Filter.” In: *Formal Methods and Software Engineering*. Ed. by Adrian Riesco and Min Zhang. Lecture Notes in Computer Science 13478. Cham: Springer International Publishing, 2022, pp. 227–243. ISBN: 978-3-031-17244-1. DOI: [10.1007/978-3-031-17244-1_14](https://doi.org/10.1007/978-3-031-17244-1_14) (cit. on pp. 34, 38).
- [KKG23] Paul Kogel, Verena Klös, and Sabine Glesner. “Learning Mealy Machines with Local Timers.” In: *Formal Methods and Software Engineering*. Ed. by Yi Li and Sofiène Tahar. Lecture Notes in Computer Science 14308. Singapore: Springer Nature, 2023, pp. 47–64. ISBN: 978-981-99-7584-6. DOI: [10.1007/978-981-99-7584-6_4](https://doi.org/10.1007/978-981-99-7584-6_4) (cit. on pp. 44, 46).
- [KKG24a] Julian Klein, Paul Kogel, and Sabine Glesner. “Efficient State Estimation of Discrete-Timed Automata.” In: *Formal Methods and Software Engineering*. Ed. by Kazuhiro Ogata, Dominique Mery, Meng Sun, and Shaoying Liu. Lecture Notes in Computer Science 15394. Singapore: Springer Nature, 2024, pp. 85–105. ISBN: 978-981-96-0617-7. DOI: [10.1007/978-981-96-0617-7_6](https://doi.org/10.1007/978-981-96-0617-7_6) (cit. on pp. 45, 46).
- [KKG24b] Julian Klein, Paul Kogel, and Sabine Glesner. “Verifying Opacity of Discrete-Timed Automata.” In: *Proceedings of the 2024 IEEE/ACM 12th International Conference on Formal Methods in Software Engineering (FormaliSE)*. New York, NY, USA: Association for Computing Machinery, June 2024, pp. 55–65. ISBN: 979-8-4007-0589-2. DOI: [10.1145/3644033.3644376](https://doi.org/10.1145/3644033.3644376) (cit. on p. 46).

- [Kle08] John C. Klensin. *Simple Mail Transfer Protocol*. Request for Comments 5321. Oct. 2008. DOI: [10.17487/RFC5321](https://doi.org/10.17487/RFC5321) (cit. on p. 33).
- [KOG19] Pavan Kantharaju, Santiago Ontañón, and Christopher W. Geib. “Scaling up CCG-Based Plan Recognition via Monte-Carlo Tree Search.” In: *2019 IEEE Conference on Games (CoG)*. Institute of Electrical and Electronics Engineers, Aug. 2019, pp. 1–8. ISBN: 978-1-7281-1884-0. DOI: [10.1109/CIG.2019.8848013](https://doi.org/10.1109/CIG.2019.8848013) (cit. on p. 37).
- [Kou+19] Erik van der Kouwe, Gernot Heiser, Dennis Andriesse, Herbert Bos, and Cristiano Giuffrida. “SoK: Benchmarking Flaws in Systems Security.” In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2019 IEEE European Symposium on Security and Privacy (EuroS&P). Institute of Electrical and Electronics Engineers, June 2019, pp. 310–325. DOI: [10.1109/EuroSP.2019.00031](https://doi.org/10.1109/EuroSP.2019.00031) (cit. on p. 12).
- [KS13] A. S. Klimowicz and V. V. Solov’ev. “Minimization of Incompletely Specified Mealy Finite-State Machines by Merging Two Internal States.” In: *Journal of Computer and Systems Sciences International* 52.3 (May 2013), pp. 400–409. ISSN: 1555-6530. DOI: [10.1134/S106423071303009X](https://doi.org/10.1134/S106423071303009X) (cit. on p. 37).
- [KS88] Mineichi Kudo and Masaru Shimbo. “Efficient Regular Grammatical Inference Techniques by the Use of Partial Similarities and Their Logical Relationships.” In: *Pattern Recognition* 21.4 (Jan. 1988), pp. 401–409. ISSN: 0031-3203. DOI: [10.1016/0031-3203\(88\)90053-2](https://doi.org/10.1016/0031-3203(88)90053-2) (cit. on p. 35).
- [KS91] L. N. Kannan and D. Sarma. “Fast Heuristic Algorithms for Finite State Machine Minimization.” In: *Proceedings of the European Conference on Design Automation*. Institute of Electrical and Electronics Engineers, Feb. 1991, pp. 192–196. DOI: [10.1109/EDAC.1991.206388](https://doi.org/10.1109/EDAC.1991.206388) (cit. on pp. 35, 37).
- [KSB17] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. “Singularity: Scientific Containers for Mobility of Compute.” In: *PLOS ONE* 12.5, e0177459 (May 2017). ISSN: 1932-6203. DOI: [10.1371/journal.pone.0177459](https://doi.org/10.1371/journal.pone.0177459) (cit. on p. 91).
- [KSG24] Paul Kogel, Wolffhardt Schwabe, and Sabine Glesner. “MMLT/ik: Efficiently Learning Mealy Machines with Local Timers by Using Imprecise Symbol Filters.” In: *Quantitative Evaluation of Systems and Formal Modeling and Analysis of Timed Systems*. Ed. by Jane Hillston, Sadegh Soudjani, and Masaki Waga. Lecture Notes in Computer Science 14996. Cham: Springer Nature Switzerland, 2024, pp. 143–159. ISBN: 978-3-031-68416-6. DOI: [10.1007/978-3-031-68416-6_9](https://doi.org/10.1007/978-3-031-68416-6_9) (cit. on pp. 44, 46).
- [KSZ17] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. “The (black) Art of Runtime Evaluation: Are We Comparing Algorithms or Implementations?” In: *Knowledge and Information Systems* 52.2 (Aug. 2017), pp. 341–378. ISSN: 0219-3116. DOI: [10.1007/s10115-016-1004-2](https://doi.org/10.1007/s10115-016-1004-2). URL: <https://doi.org/10.1007/s10115-016-1004-2> (visited on 04/20/2026) (cit. on p. 12).
- [KV94] Michael J. Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, Aug. 1994. ISBN: 978-0-262-27686-3. DOI: [10.7551/mitpress/3897.001.0001](https://doi.org/10.7551/mitpress/3897.001.0001) (cit. on pp. 32, 56).
- [Lab+23] Faezeh Labbaf, Jan Friso Groote, Hossein Hojjat, and Mohammad Reza Mousavi. “Compositional Learning for Interleaving Parallel Automata.” In: *Foundations of Software Science*

- and Computation Structures*. Ed. by Orna Kupferman and Pawel Sobocinski. Lecture Notes in Computer Science 13992. Cham: Springer Nature Switzerland, 2023, pp. 413–435. ISBN: 978-3-031-30829-1. DOI: [10.1007/978-3-031-30829-1_20](https://doi.org/10.1007/978-3-031-30829-1_20) (cit. on p. 38).
- [LAN25] LAN/MAN Standards Committee of the IEEE Computer Society. *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Std 802.11-2024. IEEE Computer Society, Apr. 28, 2025. DOI: [10.1109/IEEESTD.2025.10979691](https://doi.org/10.1109/IEEESTD.2025.10979691) (cit. on p. 44).
- [LAN99] LAN/MAN Standards Committee of the IEEE Computer Society. *Part 5: Token Ring Access Method and Physical Layer Specifications*. IEEE Std 802.11-2024. IEEE Computer Society, May 26, 1999. DOI: [10.1109/IEEESTD.1998.8941590](https://doi.org/10.1109/IEEESTD.1998.8941590) (cit. on p. 34).
- [Lan99] Kevin J. Lang. *Faster Algorithms for Finding Minimal Consistent DFAs*. Tech. rep. NEC Research Institute, Dec. 12, 1999 (cit. on p. 37).
- [LC85] Byron C. Lewis and Albert E. Crews. “The Evolution of Benchmarking as a Computer Performance Evaluation Technique.” In: *Management Information Systems Quarterly* 9.1 (Mar. 1985), pp. 7–16. ISSN: 0276-7783. DOI: [10.2307/249270](https://doi.org/10.2307/249270) (cit. on pp. 3, 12).
- [LCR11] Choonghwan Lee, Feng Chen, and Grigore Roşu. “Mining Parametric Specifications.” In: *Proceedings of the 33rd International Conference on Software Engineering*. Ed. by Richard N. Taylor, Harald Gall, and Nenad Medvidović. New York, NY, USA: Association for Computing Machinery, May 2011, pp. 591–600. ISBN: 978-1-4503-0445-0. DOI: [10.1145/1985793.1985874](https://doi.org/10.1145/1985793.1985874) (cit. on p. 43).
- [LDD08] Bernard Lambeau, Christophe Damas, and Pierre Dupont. “State-Merging DFA Induction Algorithms with Mandatory Merge Constraints.” In: *Grammatical Inference: Algorithms and Applications*. Ed. by Alexander Clark, François Coste, and Laurent Miclet. Lecture Notes in Computer Science 5278. Berlin, Heidelberg: Springer, 2008, pp. 139–153. ISBN: 978-3-540-88009-7. DOI: [10.1007/978-3-540-88009-7_11](https://doi.org/10.1007/978-3-540-88009-7_11) (cit. on pp. 35, 39).
- [Leg08] Axel Legay. “T(O)RMC: A Tool for (ω)-Regular Model Checking.” In: *Computer Aided Verification*. Ed. by Aarti Gupta and Sharad Malik. Lecture Notes in Computer Science 5123. Berlin, Heidelberg: Springer, 2008, pp. 548–551. ISBN: 9783540705451. DOI: [10.1007/978-3-540-70545-1_52](https://doi.org/10.1007/978-3-540-70545-1_52) (cit. on p. 38).
- [Leu07] Martin Leucker. “Learning Meets Verification.” In: *Formal Methods for Components and Objects*. Ed. by Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever. Lecture Notes in Computer Science 4709. Berlin, Heidelberg: Springer, 2007, pp. 127–151. ISBN: 978-3-540-74792-5. DOI: [10.1007/978-3-540-74792-5_6](https://doi.org/10.1007/978-3-540-74792-5_6) (cit. on p. 13).
- [Lin+11] Shang-Wei Lin, Étienne André, Jin Song Dong, Jun Sun, and Yang Liu. “An Efficient Algorithm for Learning Event-Recording Automata.” In: *Automated Technology for Verification and Analysis*. Ed. by Tevfik Bultan and Pao-Ann Hsiung. Lecture Notes in Computer Science 6996. Berlin, Heidelberg: Springer, 2011, pp. 463–472. ISBN: 978-3-642-24372-1. DOI: [10.1007/978-3-642-24372-1_35](https://doi.org/10.1007/978-3-642-24372-1_35) (cit. on pp. 43, 45).
- [Lin+16] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. “Log Clustering Based Problem Identification for Online Service Systems.” In: *Proceedings of the 38th International Conference on Software Engineering Companion*. New York, NY, USA:

- Association for Computing Machinery, May 2016, pp. 102–111. ISBN: 978-1-4503-4205-6. DOI: [10.1145/2889160.2889232](https://doi.org/10.1145/2889160.2889232) (cit. on pp. 67, 93).
- [Lit+17] Sascha Lity, Remo Lachmann, Malte Lochau, and Ina Schaefer. *Delta-Oriented Software Product Line Test Models – the Body Comfort System Case Study*. Informatik-Bericht 2012-07. Technische Universität Braunschweig, Apr. 21, 2017. URL: <https://github.com/TUBS-ISF/BCS-Case-Study-Full> (cit. on p. 38).
- [LLN18a] Kim G. Larsen, Florian Lorber, and Brian Nielsen. “20 Years of UPPAAL Enabled Industrial Model-Based Validation and Beyond.” In: *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*. Ed. by Tiziana Margaria and Bernhard Steffen. Lecture Notes in Computer Science 11247. Cham: Springer International Publishing, 2018, pp. 212–229. ISBN: 978-3-030-03427-6. DOI: [10.1007/978-3-030-03427-6_18](https://doi.org/10.1007/978-3-030-03427-6_18) (cit. on pp. 5, 64).
- [LLN18b] Kim Guldstrand Larsen, Florian Lorber, and Brian Nielsen. “20 Years of Real Time Model Validation.” In: *Formal Methods*. Ed. by Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik de Vink. Lecture Notes in Computer Science 10951. Cham: Springer International Publishing, 2018, pp. 22–36. ISBN: 978-3-319-95582-7. DOI: [10.1007/978-3-319-95582-7_2](https://doi.org/10.1007/978-3-319-95582-7_2) (cit. on pp. 5, 64).
- [LMP08] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè. “Automatic Generation of Software Behavioral Models.” In: *Proceedings of the 30th International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, May 2008, pp. 501–510. ISBN: 978-1-60558-079-1. DOI: [10.1145/1368088.1368157](https://doi.org/10.1145/1368088.1368157) (cit. on p. 43).
- [LN12] Martin Leucker and Daniel Neider. “Learning Minimal Deterministic Automata from Inexperienced Teachers.” In: *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*. Ed. by Tiziana Margaria and Bernhard Steffen. Lecture Notes in Computer Science 7609. Berlin, Heidelberg: Springer, 2012, pp. 524–538. ISBN: 978-3-642-34026-0. DOI: [10.1007/978-3-642-34026-0_39](https://doi.org/10.1007/978-3-642-34026-0_39) (cit. on p. 32).
- [LPP98] 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: *Grammatical Inference*. Ed. by Vasant Honavar and Giora Slutzki. Lecture Notes in Computer Science 1433. Berlin, Heidelberg: Springer, 1998, pp. 1–12. ISBN: 978-3-540-68707-8. DOI: [10.1007/BFb0054059](https://doi.org/10.1007/BFb0054059) (cit. on pp. 37, 39, 40, 64, 93).
- [LR05] Simon M. Lucas and T. Jeff Reynolds. “Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.7 (July 2005), pp. 1063–1074. ISSN: 1939-3539. DOI: [10.1109/TPAMI.2005.143](https://doi.org/10.1109/TPAMI.2005.143) (cit. on p. 39).
- [Lut+26] Simon Lutz, Daniil Kaminskyi, Florian Wittbold, Simon Dierl, Falk Howar, Barbara König, Emmanuel Müller, and Daniel Neider. “Unsupervised Automata Learning via Discrete Optimization.” In: *Logics in Artificial Intelligence*. Ed. by Giovanni Casini, Besik Dundua, and Temur Kutsia. Cham: Springer Nature Switzerland, 2026, pp. 135–153. ISBN: 978-3-032-04587-4. DOI: [10.1007/978-3-032-04587-4_9](https://doi.org/10.1007/978-3-032-04587-4_9) (cit. on pp. vii, 37).
- [LY06] Chris M. Lonvick and Tatu Ylonen. *The Secure Shell (SSH) Protocol Architecture*. Request for Comments 4251. Jan. 2006. 30 pp. DOI: [10.17487/RFC4251](https://doi.org/10.17487/RFC4251) (cit. on p. 44).

- [LY94] David Lee and Mihalis Yannakakis. “Testing Finite-State Machines: State Identification and Verification.” In: *IEEE Transactions on Computers* 43.3 (Mar. 1994), pp. 306–320. ISSN: 1557-9956. DOI: [10.1109/12.272431](https://doi.org/10.1109/12.272431) (cit. on p. 32).
- [Mai+11] Alexander Maier, Oliver Niggemann, Roman Just, Michael Jäger, and Asmir Vodenčarević. “Anomaly Detection in Production Plants Using Timed Automata. Automated Learning of Models from Observations.” In: *Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics*. Vol. 2. 2011, pp. 363–369. ISBN: 978-989-8425-74-4. DOI: [10.5220/0003538903630369](https://doi.org/10.5220/0003538903630369) (cit. on p. 43).
- [Mai14] Alexander Maier. “Online Passive Learning of Timed Automata for Cyber-Physical Production Systems.” In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. Institute of Electrical and Electronics Engineers, July 2014, pp. 60–66. ISBN: 978-1-4799-4905-2. DOI: [10.1109/INDIN.2014.6945484](https://doi.org/10.1109/INDIN.2014.6945484) (cit. on p. 43).
- [Mao+16] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. “Learning Deterministic Probabilistic Automata from a Model Checking Perspective.” In: *Machine Learning* 105.2 (Nov. 2016), pp. 255–299. ISSN: 1573-0565. DOI: [10.1007/s10994-016-5565-9](https://doi.org/10.1007/s10994-016-5565-9) (cit. on pp. 46, 69, 71, 73).
- [Mar+04] Tiziana Margaria, Oliver Niese, Harald Raffelt, and Bernhard Steffen. “Efficient Test-Based Model Generation for Legacy Reactive Systems.” In: *Proceedings. Ninth IEEE International High-Level Design Validation and Test Workshop*. Institute of Electrical and Electronics Engineers, Nov. 2004, pp. 95–100. ISBN: 0-7803-8714-7. DOI: [10.1109/HLDVT.2004.1431246](https://doi.org/10.1109/HLDVT.2004.1431246) (cit. on p. 34).
- [MCR18] Chris McMahon Stone, Tom Chothia, and Joeri de Ruiter. “Extending Automated Protocol State Learning for the 802.11 4-Way Handshake.” In: *Computer Security*. Ed. by Javier Lopez, Jianying Zhou, and Miguel Soriano. Lecture Notes in Computer Science 11098. Cham: Springer International Publishing, 2018, pp. 325–345. ISBN: 978-3-319-99073-6. DOI: [10.1007/978-3-319-99073-6_16](https://doi.org/10.1007/978-3-319-99073-6_16) (cit. on p. 44).
- [MCS12] André de Matos Pedro, Paul Andrew Crocker, and Simão Melo de Sousa. “Learning Stochastic Timed Automata from Sample Executions.” In: *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*. Ed. by Tiziana Margaria and Bernhard Steffen. Lecture Notes in Computer Science 7609. Berlin, Heidelberg: Springer, 2012, pp. 508–523. ISBN: 978-3-642-34026-0. DOI: [10.1007/978-3-642-34026-0_38](https://doi.org/10.1007/978-3-642-34026-0_38) (cit. on p. 43).
- [Mea55] George H. Mealy. “A Method for Synthesizing Sequential Circuits.” In: *The Bell System Technical Journal* 34.5 (Sept. 1955), pp. 1045–1079. ISSN: 0005-8580. DOI: [10.1002/j.1538-7305.1955.tb03788.x](https://doi.org/10.1002/j.1538-7305.1955.tb03788.x) (cit. on pp. 63, 93).
- [Men+25] Jan Mendling, Henrik Leopold, Henning Meyerhenke, and Benoît Depaire. “Methodology of Algorithm Engineering.” In: *ACM Computing Surveys* 58.4, 94 (Oct. 2025). ISSN: 0360-0300. DOI: [10.1145/3769071](https://doi.org/10.1145/3769071) (cit. on p. 12).
- [MHD22] Malte Mues, Falk Howar, and Simon Dierl. “SPouT: Symbolic Path Recording during Testing - a Concolic Executor For the JVM.” In: *Software Engineering and Formal Methods*. Ed. by Bernd-Holger Schlingloff and Ming Chai. Lecture Notes in Computer Science 13550. Cham: Springer International Publishing, 2022, pp. 91–107. ISBN: 978-3-031-17108-6. DOI: [10.1007/978-3-031-17108-6_6](https://doi.org/10.1007/978-3-031-17108-6_6) (cit. on p. vii).

- [MHS20] Matthew B. Miles, A. Michael Huberman, and Johnny Saldaña. *Qualitative Data Analysis. A Methods Sourcebook*. 4th ed. SAGE, 2020. ISBN: 978-1-5063-5307-4 (cit. on p. 76).
- [Mic25] Microsoft Corporation. *Server Message Block (SMB) Protocol Versions 2 and 3*. Open Specification MS-SMB2. Microsoft Corporation, July 28, 2025. URL: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-smb2 (cit. on p. 33).
- [Mic80] Laurent Miclet. “Regular Inference with a Tail-Clustering Method.” In: *IEEE Transactions on Systems, Man, and Cybernetics* 10.11 (Nov. 1980), pp. 737–743. ISSN: 2168-2909. DOI: [10.1109/TSMC.1980.4308394](https://doi.org/10.1109/TSMC.1980.4308394) (cit. on p. 35).
- [Mit15] Michael Mitzenmacher. “Theory without Experiments: Have We Gone Too Far?” In: *Communications of the ACM* 58.9 (Aug. 2015), pp. 40–42. ISSN: 0001-0782. DOI: [10.1145/2699413](https://doi.org/10.1145/2699413) (cit. on p. 12).
- [MM15] Irini-Eleftheria Mens and Oded Maler. “Learning Regular Languages Over Large Ordered Alphabets.” In: *Logical Methods in Computer Science* 11.3, 13 (Sept. 2015). ISSN: 1860-5974. DOI: [10.2168/LMCS-11\(3:13\)2015](https://doi.org/10.2168/LMCS-11(3:13)2015) (cit. on p. 43).
- [Moe+17] Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michał Szynwelski. “Learning Nominal Automata.” In: *ACM SIGPLAN Notices* 52.1 (Jan. 2017), pp. 613–625. ISSN: 0362-1340. DOI: [10.1145/3093333.3009879](https://doi.org/10.1145/3093333.3009879) (cit. on p. 46).
- [Moe19] Joshua Moerman. “Learning Product Automata.” In: *Proceedings of the 14th International Conference on Grammatical Inference 2018*. Ed. by Olgierd Unold, Witold Dyrka, and Wojciech Wieczorek. Proceedings of Machine Learning Research 93. PMLR, Feb. 2019, pp. 54–66. URL: <https://proceedings.mlr.press/v93/moerman19a.html> (cit. on pp. 30, 36, 38).
- [Moo56] Edward F. Moore. “Gedanken-Experiments on Sequential Machines.” In: *Automata Studies*. Ed. by C. E. Shannon and J. McCarthy. Annals of Mathematics Studies 34. Princeton: Princeton University Press, 1956, pp. 129–154. ISBN: 978-1-4008-8261-8. DOI: [10.1515/978-1-4008-8261-8-006](https://doi.org/10.1515/978-1-4008-8261-8-006) (cit. on p. 93).
- [MP95] Oded Maler and Amir Pnueli. “On the Learnability of Infinitary Regular Sets.” In: *Information and Computation* 118.2 (May 1995), pp. 316–326. ISSN: 0890-5401. DOI: [10.1006/inco.1995.1070](https://doi.org/10.1006/inco.1995.1070) (cit. on p. 32).
- [MPS17] Leonardo Mariani, Mauro Pezzè, and Mauro Santoro. “GK-Tail+ an Efficient Approach to Learn Software Models.” In: *IEEE Transactions on Software Engineering* 43.8 (Aug. 2017), pp. 715–738. ISSN: 1939-3520. DOI: [10.1109/TSE.2016.2623623](https://doi.org/10.1109/TSE.2016.2623623) (cit. on p. 43).
- [MRT17] A. S. Murawski, S. J. Ramsay, and N. Tzevelekos. “Reachability in Pushdown Register Automata.” In: *Journal of Computer and System Sciences* 87 (Aug. 2017), pp. 58–83. ISSN: 0022-0000. DOI: [10.1016/j.jcss.2017.02.008](https://doi.org/10.1016/j.jcss.2017.02.008) (cit. on pp. 12, 26, 49, 101).
- [MS11] Karl Meinke and Muddassar A. Sindhu. “Incremental Learning-Based Testing for Reactive Systems.” In: *Tests and Proofs*. Ed. by Martin Gogolla and Burkhart Wolff. Lecture Notes in Computer Science 6706. Berlin, Heidelberg: Springer, 2011, pp. 134–151. ISBN: 978-3-642-21768-5. DOI: [10.1007/978-3-642-21768-5_11](https://doi.org/10.1007/978-3-642-21768-5_11) (cit. on p. 36).
- [MS99] Faron Moller and Perdita Stevens. *The Edinburgh Concurrency Workbench User Manual (Version 7.1)*. Laboratory for Foundations of Computer Science, University of Edinburgh, July 18, 1999. URL: <https://homepages.inf.ed.ac.uk/perdita/cwb/doc/> (cit. on pp. 38, 47).

- [Mur95] Kevin P. Murphy. *Passively Learning Finite Automata*. SFI Working Paper 1996-04-017. Santa Fe Institute, Nov. 16, 1995. URL: <https://www.santafe.edu/research/results/working-papers/passively-learning-finite-automata> (cit. on p. 13).
- [Muš+24] Edi Muškardin, Martin Tappler, Bernhard K. Aichernig, and Ingo Pill. “Active Model Learning of Stochastic Reactive Systems (Extended Version).” In: *Software and Systems Modeling* 23.2 (Apr. 2024), pp. 503–524. ISSN: 1619-1374. DOI: [10.1007/s10270-024-01158-0](https://doi.org/10.1007/s10270-024-01158-0) (cit. on pp. 34, 36).
- [Nei+19] Daniel Neider, Rick Smetsers, Frits Vaandrager, and Harco Kuppens. “Benchmarks for Automata Learning and Conformance Testing.” In: *Models, Mindsets, Meta. The What, the How, and the Why Not?* Ed. by Tiziana Margaria, Susanne Graf, and Kim G. Larsen. Lecture Notes in Computer Science 11200. Cham: Springer International Publishing, 2019, pp. 390–416. ISBN: 978-3-030-22348-9. DOI: [10.1007/978-3-030-22348-9_23](https://doi.org/10.1007/978-3-030-22348-9_23) (cit. on pp. 5, 7, 14, 15, 27, 37, 38, 47, 51, 57, 75, 91, 99).
- [Nic14] Cyril Nicaud. “Random Deterministic Automata.” In: *Mathematical Foundations of Computer Science 2014*. Ed. by Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik. Lecture Notes in Computer Science 8634. Berlin, Heidelberg: Springer, 2014, pp. 5–23. ISBN: 978-3-662-44522-8. DOI: [10.1007/978-3-662-44522-8_2](https://doi.org/10.1007/978-3-662-44522-8_2) (cit. on pp. 7, 14).
- [Nie03] Oliver Niese. “An Integrated Approach to Testing Complex Systems.” PhD thesis. Universität Dortmund, Dec. 2003. DOI: [10.17877/DE290R-14871](https://doi.org/10.17877/DE290R-14871) (cit. on p. 34).
- [NJ13] Daniel Neider and Nils Jansen. “Regular Model Checking Using Solver Technologies and Automata Learning.” In: *NASA Formal Methods*. Ed. by Guillaume Brat, Neha Rungta, and Arnaud Venet. Lecture Notes in Computer Science 7871. Berlin, Heidelberg: Springer, 2013, pp. 16–31. ISBN: 978-3-642-38088-4. DOI: [10.1007/978-3-642-38088-4_2](https://doi.org/10.1007/978-3-642-38088-4_2) (cit. on pp. 34, 36, 38, 41, 45).
- [NSV04] Frank Neven, Thomas Schwentick, and Victor Vianu. “Finite State Machines for Strings Over Infinite Alphabets.” In: *ACM Transactions on Computational Logic* 5.3 (July 2004), pp. 403–435. ISSN: 1529-3785. DOI: [10.1145/1013560.1013562](https://doi.org/10.1145/1013560.1013562) (cit. on pp. 12, 26).
- [OAS19] OASIS Message Queuing Telemetry Transport (MQTT) TC. *MQTT Version 5.0*. OASIS Standard. OASIS Open, Mar. 7, 2019. URL: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> (cit. on pp. 33, 34, 44).
- [OG93] José Oncina and Pedro García. “Identifying Regular Languages in Polynomial Time.” In: *Advances in Structural and Syntactic Pattern Recognition*. Ed. by Horst Bunke. Series in Machine Perception and Artificial Intelligence 5. World Scientific, Feb. 1993, pp. 99–108. ISBN: 978-981-4505-15-4. DOI: [10.1142/9789812797919_0007](https://doi.org/10.1142/9789812797919_0007). (Visited on 05/26/2025) (cit. on pp. 35, 63).
- [OGV93] José Oncina, Pedro Garcia, and Enrique Vidal. “Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.5 (May 1993), pp. 448–458. ISSN: 1939-3539. DOI: [10.1109/34.211465](https://doi.org/10.1109/34.211465) (cit. on pp. 30, 35).
- [Ohm+14] Tony Ohmann, Michael Herzberg, Sebastian Fiss, Armand Halbert, Marc Palyart, Ivan Beschastnikh, and Yuriy Brun. “Behavioral Resource-Aware Model Inference.” In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*.

- New York, NY, USA: Association for Computing Machinery, Sept. 2014, pp. 19–30. ISBN: 978-1-4503-3013-8. DOI: [10.1145/2642937.2642988](https://doi.org/10.1145/2642937.2642988) (cit. on pp. 43, 46).
- [Onc98] José Oncina. “The Data Driven Approach Applied to the OSTIA Algorithm.” In: *Grammatical Inference*. Ed. by Vasant Honavar and Giora Slutzki. Lecture Notes in Computer Science 1433. Berlin, Heidelberg: Springer, 1998, pp. 50–56. ISBN: 978-3-540-68707-8. DOI: [10.1007/BFb0054063](https://doi.org/10.1007/BFb0054063) (cit. on p. 37).
- [OS07] Adam Oliner and Jon Stearley. “What Supercomputers Say: A Study of Five System Logs.” In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)*. Institute of Electrical and Electronics Engineers, June 2007, pp. 575–584. ISBN: 0-7695-2855-4. DOI: [10.1109/DSN.2007.103](https://doi.org/10.1109/DSN.2007.103) (cit. on pp. 37, 38).
- [OS98] A. L. Oliveira and J. P. M. Silva. “Efficient Search Techniques for the Inference of Minimum Size Finite Automata.” In: *String Processing and Information Retrieval: A South American Symposium*. Institute of Electrical and Electronics Engineers, Sept. 1998, pp. 81–89. ISBN: 0-8186-8664-2. DOI: [10.1109/SPIRE.1998.712986](https://doi.org/10.1109/SPIRE.1998.712986) (cit. on pp. 37, 39).
- [OV96] José Oncina and Miguel Angel Varó. “Using Domain Information during the Learning of a Subsequential Transducer.” In: *Grammatical Interference: Learning Syntax from Sentences*. Ed. by Laurent Miclet and Colin de la Higuera. Lecture Notes in Computer Science 1147. Berlin, Heidelberg: Springer, 1996, pp. 301–312. ISBN: 978-3-540-70678-6. DOI: [10.1007/BFb0033364](https://doi.org/10.1007/BFb0033364) (cit. on pp. 30, 32).
- [OWA25] OWASP Foundation. *OWASP Benchmark*. 2025. URL: <https://owasp.org/www-project-benchmark/> (cit. on p. 14).
- [PA21] Andrea Pferscher and Bernhard K. Aichernig. “Fingerprinting Bluetooth Low Energy Devices via Active Automata Learning.” In: *Formal Methods*. Ed. by Marieke Huisman, Corina Păsăreanu, and Naijun Zhan. Lecture Notes in Computer Science 13047. Cham: Springer International Publishing, 2021, pp. 524–542. ISBN: 978-3-030-90870-6. DOI: [10.1007/978-3-030-90870-6_28](https://doi.org/10.1007/978-3-030-90870-6_28) (cit. on p. 44).
- [PA22] Andrea Pferscher and Bernhard K. Aichernig. “Stateful Black-Box Fuzzing of Bluetooth Devices Using Automata Learning.” In: *NASA Formal Methods*. Ed. by Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez. Lecture Notes in Computer Science 13260. Cham: Springer International Publishing, 2022, pp. 373–392. ISBN: 978-3-031-06773-0. DOI: [10.1007/978-3-031-06773-0_20](https://doi.org/10.1007/978-3-031-06773-0_20) (cit. on p. 44).
- [Pas+22] Fabrizio Pastore, Daniela Micucci, Michell Guzman, and Leonardo Mariani. “TkT: Automatic Inference of Timed and Extended Pushdown Automata.” In: *IEEE Transactions on Software Engineering* 48.2 (Feb. 2022), pp. 617–636. ISSN: 1939-3520. DOI: [10.1109/TSE.2020.2998527](https://doi.org/10.1109/TSE.2020.2998527) (cit. on pp. 43, 65).
- [Pet+14] Alexandre Petrenko, Keqin Li, Roland Groz, Karim Hossen, and Catherine Oriat. “Inferring Approximated Models for Systems Engineering.” In: *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*. 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering. Ed. by Peter J. Clarke, Mohammad Zulkernine, Raymond A. Paul, and Jie Xu. Institute of Electrical and Electronics Engineers, Jan. 2014, pp. 249–253. ISBN: 978-1-4799-3466-9. DOI: [10.1109/HASE.2014.46](https://doi.org/10.1109/HASE.2014.46) (cit. on pp. 34, 40).

- [Pet+19] Alexandre Petrenko, Florent Avellaneda, Roland Groz, and Catherine Oriat. “FSM Inference and Checking Sequence Construction Are Two Sides of the Same Coin.” In: *Software Quality Journal* 27.2 (June 2019), pp. 651–674. ISSN: 1573-1367. DOI: [10.1007/s11219-018-9429-3](https://doi.org/10.1007/s11219-018-9429-3) (cit. on pp. 37, 39).
- [PF91] Sara Porat and Jerome A. Feldman. “Learning Automata from Ordered Examples.” In: *Machine Learning* 7.2 (Sept. 1991), pp. 109–138. ISSN: 1573-0565. DOI: [10.1007/BF00114841](https://doi.org/10.1007/BF00114841) (cit. on p. 32).
- [Pfl73] C. P. Pfleeger. “State Reduction in Incompletely Specified Finite-State Machines.” In: *IEEE Transactions on Computers* C-22.12 (Dec. 1973), pp. 1099–1102. ISSN: 1557-9956. DOI: [10.1109/T-C.1973.223655](https://doi.org/10.1109/T-C.1973.223655) (cit. on p. 32).
- [PG93] Ruchir Puri and Jun Gu. “An Efficient Algorithm to Search for Minimal Closed Covers in Sequential Machines.” In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 12.6 (June 1993), pp. 737–745. ISSN: 1937-4151. DOI: [10.1109/43.229748](https://doi.org/10.1109/43.229748) (cit. on pp. 35, 37).
- [Pin+24] Romain Piquié, Lionel Roucoules, Pierre-Alain Yvars, and Raphaël Chenouard. “Operationalizing Community-Based Open Scientific Design Research Benchmarks: Application to Model-Based Architecture Design Synthesis.” In: *Proceedings of the Design Society* 4 (2024), pp. 95–104. DOI: [10.1017/pds.2024.12](https://doi.org/10.1017/pds.2024.12) (cit. on p. 12).
- [Pit89] Leonard Pitt. “Inductive Inference, DFAs, and Computational Complexity.” In: *Analogical and Inductive Inference*. Ed. by Klaus P. Jantke. Lecture Notes in Computer Science 397. Berlin, Heidelberg: Springer, 1989, pp. 18–44. ISBN: 978-3-540-46798-4. DOI: [10.1007/3-540-51734-0_50](https://doi.org/10.1007/3-540-51734-0_50) (cit. on p. 13).
- [PNH98] Rajesh Parekh, Codrin Nichitui, and Vasant Honavar. “A Polynomial Time Incremental Algorithm for Learning DFA.” In: *Grammatical Inference*. Ed. by Vasant Honavar and Giora Slutzki. Lecture Notes in Computer Science 1433. Berlin, Heidelberg: Springer, 1998, pp. 37–49. ISBN: 978-3-540-68707-8. DOI: [10.1007/BFb0054062](https://doi.org/10.1007/BFb0054062) (cit. on p. 32).
- [PO99] Jorge M. Pena and Arlindo L. Oliveira. “A New Algorithm for Exact Reduction of Incompletely Specified Finite State Machines.” In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18.11 (Nov. 1999), pp. 1619–1632. ISSN: 1937-4151. DOI: [10.1109/43.806807](https://doi.org/10.1109/43.806807) (cit. on p. 37).
- [Pop02] Karl Popper. *The Logic of Scientific Discovery*. 2nd ed. Routledge Classics. London, New York: Routledge, Feb. 21, 2002. ISBN: 978-0-203-99462-7. URL: <https://routledge.com/9780415278447> (cit. on p. 3).
- [PPV00] Dewayne E. Perry, Adam A. Porter, and Lawrence G. Votta. “Empirical Studies of Software Engineering: A Roadmap.” In: *ICSE’00: Proceedings of the Conference on The Future of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2000, pp. 345–355. ISBN: 978-1-58113-253-3. DOI: [10.1145/336512.336586](https://doi.org/10.1145/336512.336586) (cit. on p. 11).
- [PS89] Ian Pressman and David Singmaster. “The Jealous Husbands and the Missionaries and Cannibals.” In: *The Mathematical Gazette* 73.464 (1989), pp. 73–81. DOI: [10.2307/3619658](https://doi.org/10.2307/3619658) (cit. on p. 53).
- [PU59] M. C. Paull and S. H. Unger. “Minimizing the Number of States in Incompletely Specified Sequential Switching Functions.” In: *IRE Transactions on Electronic Computers* EC-8.3

- (Sept. 1959), pp. 356–367. ISSN: 0367-9950. DOI: [10.1109/TEC.1959.5222697](https://doi.org/10.1109/TEC.1959.5222697) (cit. on pp. 32, 65).
- [PVY99] Doron Peled, Moshe Y. Vardi, and Mihalis Yannakakis. “Black Box Checking.” In: *Formal Methods for Protocol Engineering and Distributed Systems*. Ed. by Jianping Wu, Samuel T. Chanson, and Qiang Gao. IFIP Advances in Information and Communication Technology 28. Boston, MA: Springer US, 1999, pp. 225–240. ISBN: 978-0-387-35578-8. DOI: [10.1007/978-0-387-35578-8_13](https://doi.org/10.1007/978-0-387-35578-8_13) (cit. on p. 2).
- [Rai06] Lesley Anne Raitt. “Random Generation of Finite Automata Over the Domain of the Regular Languages.” MA thesis. University of Stellenbosch, Dec. 2006. URL: <https://hdl.handle.net/10019.1/19646> (cit. on p. 14).
- [RAR19] Abdullah Rasool, Greg Alpár, and Joeri de Ruiter. *State Machine Inference of QUIC*. arXiv Preprint. Mar. 2019. DOI: [10.48550/arXiv.1903.04384](https://doi.org/10.48550/arXiv.1903.04384) (cit. on p. 44).
- [Reb67] Arthur S. Reber. “Implicit Learning of Artificial Grammars.” In: *Journal of Verbal Learning and Verbal Behavior* 6.6 (Dec. 1967), pp. 855–863. ISSN: 0022-5371. DOI: [10.1016/S0022-5371\(67\)80149-X](https://doi.org/10.1016/S0022-5371(67)80149-X) (cit. on p. 37).
- [Res18] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Request for Comments 8446. Aug. 2018. 160 pp. DOI: [10.17487/RFC8446](https://doi.org/10.17487/RFC8446) (cit. on p. 44).
- [RH09] Per Runeson and Martin Höst. “Guidelines for Conducting and Reporting Case Study Research in Software Engineering.” In: *Empirical Software Engineering* 14.2 (Apr. 2009), pp. 131–164. ISSN: 1573-7616. DOI: [10.1007/s10664-008-9102-8](https://doi.org/10.1007/s10664-008-9102-8) (cit. on p. 4).
- [Ron95] Dana Ron. “Automata Learning and Its Applications.” PhD thesis. Hebrew University, 1995 (cit. on pp. 33, 37).
- [RP15] Joeri de Ruiter and Erik Poll. “Protocol State Fuzzing of TLS Implementations.” In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 193–206. ISBN: 978-1-939133-11-3. URL: <https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter> (cit. on p. 44).
- [RP97] Anand Raman and John Patrick. “The sk-strings Method for Inferring PFSA.” In: *Proceedings of the Workshop on Automata Induction, Grammatical Inference and Language Acquisition at the 14th International Conference on Machine Learning (ICML97)*, 1997 (cit. on p. 39).
- [RR01] Steven P. Reiss and Manos Renieris. “Encoding Program Executions.” In: *The 23rd International Conference on Software Engineering. ICSE 2001*. Institute of Electrical and Electronics Engineers, May 2001, pp. 221–230. ISBN: 0-7695-1050-7. DOI: [10.1109/ICSE.2001.919096](https://doi.org/10.1109/ICSE.2001.919096) (cit. on p. 33).
- [RS93] Ronald L. Rivest and Robert E. Schapire. “Inference of Finite Automata Using Homing Sequences.” In: *Information and Computation* 103.2 (Apr. 1993), pp. 299–347. ISSN: 0890-5401. DOI: [10.1006/inco.1993.1021](https://doi.org/10.1006/inco.1993.1021) (cit. on p. 36).
- [RS94] Ronald L. Rivest and Robert E. Schapire. “Diversity-Based Inference of Finite Automata.” In: *Journal of the ACM* 41.3 (May 1994), pp. 555–589. ISSN: 0004-5411. DOI: [10.1145/176584.176589](https://doi.org/10.1145/176584.176589) (cit. on p. 36).

- [RTM22] Eric Rescorla, Hannes Tschofenig, and Nagenia Modadugu. *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*. Request for Comments 9147. Apr. 21, 2022. DOI: [10.17487/RFC9147](https://doi.org/10.17487/RFC9147) (cit. on p. 44).
- [Rub+97] Allan Rubens, Steve Willens, Carl Rigney, and William A. Simpson. *Remote Authentication Dial In User Service (RADIUS)*. Request for Comments 2058. Jan. 1997. DOI: [10.17487/RFC2058](https://doi.org/10.17487/RFC2058) (cit. on p. 43).
- [Run+12] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering. Guidelines and Examples*. Hoboken, New Jersey: John Wiley & Sons, Ltd, 2012. ISBN: 978-1-118-18103-4. DOI: [10.1002/9781118181034](https://doi.org/10.1002/9781118181034) (cit. on pp. 4, 11).
- [RV84] M. Richetin and F. Vernadat. “Efficient Regular Grammatical Inference for Pattern Recognition.” In: *Pattern Recognition* 17.2 (Jan. 1984), pp. 245–250. ISSN: 0031-3203. DOI: [10.1016/0031-3203\(84\)90063-3](https://doi.org/10.1016/0031-3203(84)90063-3) (cit. on pp. 33, 35).
- [Sag+23] Konstantinos Sagonas, Bengt Jonsson, Falk Howar, Simon Dierl, Paul Fiterau-Brostean, and Fredrik Tåkvist. *Reproduction Artifact for TACAS 2024 Paper “Scalable Tree-Based Register Automata Learning”*. Dec. 29, 2023. DOI: [10.5281/zenodo.10442556](https://doi.org/10.5281/zenodo.10442556) (cit. on p. 95).
- [Sai11] Peter Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. Request for Comments 6120. RFC Editor, Mar. 2011. DOI: [10.17487/RFC6120](https://doi.org/10.17487/RFC6120) (cit. on pp. 1, 17).
- [SBH25] Mahboubeh Samadi, Aryan Bastany, and Hossein Hojjat. “Compositional Learning for Synchronous Parallel Automata.” In: *Fundamental Approaches to Software Engineering*. Ed. by Artur Boronat and Gordon Fraser. Lecture Notes in Computer Science 15693. Cham: Springer Nature Switzerland, 2025, pp. 101–121. ISBN: 978-3-031-90900-9. DOI: [10.1007/978-3-031-90900-9_6](https://doi.org/10.1007/978-3-031-90900-9_6) (cit. on pp. 38, 40).
- [SCC02] William R. Shadish, Thomas D. Cook, and Donald T. Campbell. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Boston, New York: Houghton Mifflin Company, 2002. ISBN: 978-0-395-61556-0 (cit. on p. 3).
- [Sch+02] Eve Schooler, Jonathan Rosenberg, Henning Schulzrinne, Alan Johnston, Gonzalo Camarillo, Jon Peterson, Robert Sparks, and Mark J. Handley. *SIP: Session Initiation Protocol*. Request for Comments 3261. RFC Editor, July 2002. DOI: [10.17487/RFC3261](https://doi.org/10.17487/RFC3261) (cit. on pp. 33, 34, 44, 53).
- [Sch+25] Leon Schwarzer, Matthias Zeller, Daniel Casado Herraiez, Simon Dierl, Michael Heidingsfeld, and Cyrill Stachniss. “Self-Supervised Moving Object Segmentation of Sparse and Noisy Radar Point Clouds.” In: *2025 IEEE 28th International Conference on Intelligent Transportation Systems (ITSC)*. Institute of Electrical and Electronics Engineers, 2025, pp. 276–283. ISBN: 979-8-3315-2418-0. DOI: [10.1109/ITSC60802.2025.11423257](https://doi.org/10.1109/ITSC60802.2025.11423257) (cit. on p. vii).
- [Sea99] Carolyn B. Seaman. “Qualitative Methods in Empirical Studies of Software Engineering.” In: *IEEE Transactions on Software Engineering* 25.4 (July 1999), pp. 557–572. ISSN: 1939-3520. DOI: [10.1109/32.799955](https://doi.org/10.1109/32.799955) (cit. on p. 76).
- [See+06] Carolyn C. Seepersad, Kjartan Pedersen, Jan Emblemståg, Reid Bailey, Janet K. Allen, and Farrokh Mistree. “The Validation Square: How Does One Verify and Validate a Design Method?” In: *Decision Making in Engineering Design*. Ed. by Kemper E. Lewis,

- Wei Chen, and Linda C. Schmidt. ASME Press, 2006. ISBN: 978-0-7918-0246-5. DOI: [10.1115/1.802469.ch25](https://doi.org/10.1115/1.802469.ch25) (cit. on p. 11).
- [Seg06] Luc Segoufin. “Automata and Logics for Words and Trees Over an Infinite Alphabet.” In: *Computer Science Logic*. Ed. by Zoltán Ésik. Lecture Notes in Computer Science 4207. Berlin, Heidelberg: Springer, 2006, pp. 41–57. ISBN: 978-3-540-45459-5. DOI: [10.1007/11874683_3](https://doi.org/10.1007/11874683_3) (cit. on pp. 12, 13, 26).
- [SEH03] Susan Elliott Sim, Steve Easterbrook, and Richard C. Holt. “Using Benchmarking to Advance Research: A Challenge to Software Engineering.” In: *25th International Conference on Software Engineering, 2003. Proceedings*. Institute of Electrical and Electronics Engineers, May 2003, pp. 74–83. ISBN: 0-7695-1877-X. DOI: [10.1109/ICSE.2003.1201189](https://doi.org/10.1109/ICSE.2003.1201189) (cit. on p. 11).
- [SF18] Klaas-Jan Stol and Brian Fitzgerald. “The ABC of Software Engineering Research.” In: *ACM Transactions on Software Engineering and Methodology* 27.3, 11 (Sept. 2018). ISSN: 1049-331X. DOI: [10.1145/3241743](https://doi.org/10.1145/3241743) (cit. on p. 3, 11).
- [SFV18] Rick Smetsers, Paul Fiterău-Broștean, and Frits Vaandrager. “Model Learning As a Satisfiability Modulo Theories Problem.” In: *Language and Automata Theory and Applications*. Ed. by Shmuel Tomi Klein, Carlos Martín-Vide, and Dana Shapira. Lecture Notes in Computer Science 10792. Cham: Springer International Publishing, 2018, pp. 182–194. ISBN: 978-3-319-77313-1. DOI: [10.1007/978-3-319-77313-1_14](https://doi.org/10.1007/978-3-319-77313-1_14) (cit. on p. 47).
- [SG09] Muzammil Shahbaz and Roland Groz. “Inferring Mealy Machines.” In: *FM 2009: Formal Methods*. Ed. by Ana Cavalcanti and Dennis R. Dams. Lecture Notes in Computer Science 5850. Berlin, Heidelberg: Springer, 2009, pp. 207–222. ISBN: 978-3-642-05089-3. DOI: [10.1007/978-3-642-05089-3_14](https://doi.org/10.1007/978-3-642-05089-3_14) (cit. on pp. 38, 40).
- [SG14] Muzammil Shahbaz and Roland Groz. “Analysis and Testing of Black-Box Component-Based Systems by Inferring Partial Models.” In: *Software Testing, Verification and Reliability* 24.4 (2014), pp. 253–288. ISSN: 1099-1689. DOI: [10.1002/stvr.1491](https://doi.org/10.1002/stvr.1491) (cit. on p. 34).
- [SGL95] J. M. Sánchez, A. O. Garnica, and J. Lanchares. “A Genetic Algorithm for Reducing the Number of States in Incompletely Specified Finite State Machines.” In: *Microelectronics Journal* 26.5 (July 1995), pp. 463–470. ISSN: 1879-2391. DOI: [10.1016/0026-2692\(95\)98948-Q](https://doi.org/10.1016/0026-2692(95)98948-Q) (cit. on p. 37).
- [Sha03] Mary Shaw. “Writing Good Software Engineering Research Papers.” In: *25th International Conference on Software Engineering, 2003. Proceedings*. Institute of Electrical and Electronics Engineers, May 2003, pp. 726–736. ISBN: 0-7695-1877-X. DOI: [10.1109/ICSE.2003.1201262](https://doi.org/10.1109/ICSE.2003.1201262) (cit. on pp. 4, 11).
- [SHM11] Bernhard Steffen, Falk Howar, and Maik Merten. “Introduction to Active Automata Learning from a Practical Perspective.” In: *Formal Methods for Eternal Networked Software Systems*. Ed. by Marco Bernardo and Valérie Issarny. Lecture Notes in Computer Science 6659. Berlin, Heidelberg: Springer, 2011, pp. 256–296. ISBN: 978-3-642-21455-4. DOI: [10.1007/978-3-642-21455-4_8](https://doi.org/10.1007/978-3-642-21455-4_8) (cit. on p. 36).
- [Shr+20] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. “ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks.” In: *2020 IEEE/CVF Conference on Computer*

- Vision and Pattern Recognition (CVPR)*. Institute of Electrical and Electronics Engineers, June 2020, pp. 10737–10746. ISBN: 978-1-7281-7168-5. DOI: [10.1109/CVPR42600.2020.01075](https://doi.org/10.1109/CVPR42600.2020.01075) (cit. on pp. 37, 41, 93).
- [Shv+21] Maayan Shvo, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. “Interpretable Sequence Classification via Discrete Optimization.” In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.11 (May 2021), pp. 9647–9656. ISSN: 2374-3468. DOI: [10.1609/aaai.v35i11.17161](https://doi.org/10.1609/aaai.v35i11.17161) (cit. on pp. 37, 93).
- [Sme+15] Wouter Smeenk, Joshua Moerman, Frits Vaandrager, and David N. Jansen. “Applying Automata Learning to Embedded Control Software.” In: *Formal Methods and Software Engineering*. Ed. by Michael Butler, Sylvain Conchon, and Fatiha Zaïdi. Lecture Notes in Computer Science 9407. Cham: Springer International Publishing, 2015, pp. 67–83. ISBN: 978-3-319-25423-4. DOI: [10.1007/978-3-319-25423-4_5](https://doi.org/10.1007/978-3-319-25423-4_5) (cit. on p. 44).
- [SR24] Pankaj Sharma and Saravanakumar Raju. “Metaheuristic Optimization Algorithms: A Comprehensive Overview and Classification of Benchmark Test Functions.” In: *Soft Computing* 28.4 (Feb. 2024), pp. 3123–3186. ISSN: 1433-7479. DOI: [10.1007/s00500-023-09276-5](https://doi.org/10.1007/s00500-023-09276-5) (cit. on p. 12).
- [SSE12] Muzammil Shahbaz, K. C. Shashidhar, and Robert Eschbach. “Specification Inference Using Systematic Reverse-Engineering Methodologies: An Automotive Industry Application.” In: *IEEE Software* 29.6 (Nov. 2012), pp. 62–69. ISSN: 1937-4194. DOI: [10.1109/MS.2011.159](https://doi.org/10.1109/MS.2011.159) (cit. on p. 34).
- [STN22] Randall R. Stewart, Michael Tüxen, and Karen E. E. Nielsen. *Stream Control Transmission Protocol*. Request for Comments 9260. June 2022. 133 pp. DOI: [10.17487/RFC9260](https://doi.org/10.17487/RFC9260) (cit. on pp. 44, 45).
- [TAB17] 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)*. Institute of Electrical and Electronics Engineers, Mar. 2017, pp. 276–287. ISBN: 978-1-5090-6031-3. DOI: [10.1109/ICST.2017.32](https://doi.org/10.1109/ICST.2017.32) (cit. on p. 44).
- [TAL22] Martin Tappler, Bernhard K. Aichernig, and Florian Lorber. “Timed Automata Learning via SMT Solving.” In: *NASA Formal Methods*. Ed. by Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez. Lecture Notes in Computer Science 13260. Cham: Springer International Publishing, 2022, pp. 489–507. ISBN: 978-3-031-06773-0. DOI: [10.1007/978-3-031-06773-0_26](https://doi.org/10.1007/978-3-031-06773-0_26) (cit. on pp. 35, 39).
- [Tap+19] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. “Time to Learn – Learning Timed Automata from Tests.” In: *Formal Modeling and Analysis of Timed Systems*. Ed. by Étienne André and Mariëlle Stoelinga. Lecture Notes in Computer Science 11750. Cham: Springer International Publishing, 2019, pp. 216–235. ISBN: 978-3-030-29662-9. DOI: [10.1007/978-3-030-29662-9_13](https://doi.org/10.1007/978-3-030-29662-9_13) (cit. on pp. 43, 46, 48).
- [Tap+21] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. “L*-Based Learning of Markov Decision Processes (Extended Version).” In: *Formal Aspects of Computing* 33.4 (Aug. 2021), pp. 575–615. ISSN: 1433-299X. DOI: [10.1007/s00165-021-00536-5](https://doi.org/10.1007/s00165-021-00536-5) (cit. on pp. 34, 36).

- [Tap+24] 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: *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. Institute of Electrical and Electronics Engineers, May 2024, pp. 197–208. ISBN: 979-8-3503-0818-1. DOI: [10.1109/ICST60714.2024.00026](https://doi.org/10.1109/ICST60714.2024.00026) (cit. on p. 38).
- [TB22] Martin Thomson and Cory Benfield. *HTTP/2*. Request for Comments 9113. June 2022. DOI: [10.17487/RFC9113](https://doi.org/10.17487/RFC9113) (cit. on p. 43).
- [TDH00] Franck Thollard, Pierre Dupont, and Colin de la Higuera. “Probabilistic DFA Inference Using Kullback-Leibler Divergence and Minimality.” In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*. Ed. by Pat Langley. Morgan Kaufmann, 2000, pp. 975–982. ISBN: 978-1-55860-707-1 (cit. on p. 37).
- [TE11] Fedor Tsarev and Kirill Egorov. “Finite State Machine Induction Using Genetic Algorithm Based on Testing and Model Checking.” In: *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*. New York, NY, USA: Association for Computing Machinery, July 2011, pp. 759–762. ISBN: 978-1-4503-0690-4. DOI: [10.1145/2001858.2002085](https://doi.org/10.1145/2001858.2002085) (cit. on p. 35).
- [Thi+22] Jeyan Thiagalingam, Mallikarjun Shankar, Geoffrey Fox, and Tony Hey. “Scientific Machine Learning Benchmarks.” In: *Nature Reviews Physics* 4.6 (June 2022), pp. 413–420. ISSN: 2522-5820. DOI: [10.1038/s42254-022-00441-7](https://doi.org/10.1038/s42254-022-00441-7) (cit. on p. 12).
- [TIL04] Emmanuel Munguia Tapia, Stephen S. Intille, and Kent Larson. “Activity Recognition in the Home Using Simple and Ubiquitous Sensors.” In: *Pervasive Computing*. Ed. by Alois Ferscha and Friedemann Mattern. Lecture Notes in Computer Science 3001. Berlin, Heidelberg: Springer, 2004, pp. 158–175. ISBN: 978-3-540-24646-6. DOI: [10.1007/978-3-540-24646-6_10](https://doi.org/10.1007/978-3-540-24646-6_10) (cit. on pp. 37, 93).
- [Tze92] Wen-Guey Tzeng. “Learning Probabilistic Automata and Markov Chains via Queries.” In: *Machine Learning* 8.2 (Mar. 1992), pp. 151–166. ISSN: 1573-0565. DOI: [10.1023/A:1022616503659](https://doi.org/10.1023/A:1022616503659) (cit. on p. 32).
- [UBS18] Vladimir Ulyantsev, Igor Buzhinsky, and Anatoly Shalyto. “Exact Finite-State Machine Identification from Scenarios and Temporal Properties.” In: *International Journal on Software Tools for Technology Transfer* 20.1 (Feb. 2018), pp. 35–55. ISSN: 1433-2787. DOI: [10.1007/s10009-016-0442-1](https://doi.org/10.1007/s10009-016-0442-1) (cit. on pp. 33, 35, 39).
- [Ull15] Jeffrey D. Ullman. “Experiments As Research Validation: Have We Gone Too Far?” In: *Communications of the ACM* 58.9 (Aug. 2015), pp. 37–39. ISSN: 0001-0782. DOI: [10.1145/2699405](https://doi.org/10.1145/2699405) (cit. on p. 12).
- [US20] Henning Urbat and Lutz Schröder. “Automata Learning: An Algebraic Approach.” In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. New York, NY, USA: Association for Computing Machinery, July 2020, pp. 900–914. ISBN: 978-1-4503-7104-9. DOI: [10.1145/3373718.3394775](https://doi.org/10.1145/3373718.3394775) (cit. on p. 32).
- [UT12] Vladimir I. Ulyantsev and Fedor N. Tsarev. “Extended Finite-State Machine Induction Using SAT-Solver.” In: *IFAC Proceedings Volumes* 45.6 (May 2012), pp. 236–241. ISSN: 1474-6670. DOI: [10.3182/20120523-3-RO-2023.00179](https://doi.org/10.3182/20120523-3-RO-2023.00179) (cit. on pp. 46, 48).

- [UZS15] Vladimir Ulyantsev, Ilya Zakirzyanov, and Anatoly Shalyto. “BFS-Based Symmetry Breaking Predicates for DFA Identification.” In: *Language and Automata Theory and Applications*. Ed. by Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe. Lecture Notes in Computer Science 8977. Cham: Springer International Publishing, 2015, pp. 611–622. ISBN: 978-3-319-15579-1. DOI: [10.1007/978-3-319-15579-1_48](https://doi.org/10.1007/978-3-319-15579-1_48) (cit. on p. 39).
- [Vaa+22] Frits Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. “A New Approach for Active Automata Learning Based on Apartness.” In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Dana Fisman and Grigore Rosu. Lecture Notes in Computer Science 13243. Cham: Springer International Publishing, 2022, pp. 223–243. ISBN: 978-3-030-99524-9. DOI: [10.1007/978-3-030-99524-9_12](https://doi.org/10.1007/978-3-030-99524-9_12) (cit. on p. 38).
- [Vaa17] Frits Vaandrager. “Model Learning.” In: *Communications of the ACM* 60.2 (Jan. 2017), pp. 86–95. ISSN: 0001-0782. DOI: [10.1145/2967606](https://doi.org/10.1145/2967606) (cit. on p. 13).
- [VBE21] Frits Vaandrager, Roderick Bloem, and Masoud Ebrahimi. “Learning Mealy Machines with One Timer.” In: *Language and Automata Theory and Applications*. Ed. by Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron. Lecture Notes in Computer Science 12638. Cham: Springer International Publishing, 2021, pp. 157–170. ISBN: 978-3-030-68195-1. DOI: [10.1007/978-3-030-68195-1_13](https://doi.org/10.1007/978-3-030-68195-1_13) (cit. on pp. 7, 28, 44, 63).
- [VEH14] Sicco Verwer, Rémi Eyraud, and Colin de la Higuera. “PAUTOMAC: A Probabilistic Automata and Hidden Markov Models Learning Competition.” In: *Machine Learning* 96.1 (July 2014), pp. 129–154. ISSN: 1573-0565. DOI: [10.1007/s10994-013-5409-9](https://doi.org/10.1007/s10994-013-5409-9) (cit. on pp. 5, 13).
- [Vil24] Juan Miguel Vilar. “A Categorical Interpretation of State Merging Algorithms for DFA Inference.” In: *Pattern Recognition* 150, 110326 (June 2024). ISSN: 0031-3203. DOI: [10.1016/j.patcog.2024.110326](https://doi.org/10.1016/j.patcog.2024.110326) (cit. on p. 32).
- [VS25] Frits Vaandrager and Martijn Sanders. “ $L^\#$ for DFAs.” In: *Principles of Verification. Cycling the Probabilistic Landscape*. Ed. by Nils Jansen, Sebastian Junges, Benjamin Lucien Kaminski, Christoph Matheja, Thomas Noll, Tim Quatmann, Mariëlle Stoelinga, and Matthias Volk. Cham: Springer Nature Switzerland, 2025, pp. 155–172. ISBN: 978-3-031-75778-5. DOI: [10.1007/978-3-031-75778-5_8](https://doi.org/10.1007/978-3-031-75778-5_8) (cit. on pp. 38, 40, 41).
- [VWW10] Sicco Verwer, Mathijs de Weerd, and Cees Witteveen. “A Likelihood-Ratio Test for Identifying Probabilistic Deterministic Real-Time Automata from Positive Data.” In: *Grammatical Inference: Theoretical Results and Applications*. Ed. by José M. Sempere and Pedro García. Lecture Notes in Computer Science 6339. Berlin, Heidelberg: Springer, 2010, pp. 203–216. ISBN: 978-3-642-15488-1. DOI: [10.1007/978-3-642-15488-1_17](https://doi.org/10.1007/978-3-642-15488-1_17) (cit. on p. 48).
- [VWW11] Sicco Verwer, Mathijs de Weerd, and Cees Witteveen. “The Efficiency of Identifying Timed Automata and the Power of Clocks.” In: *Information and Computation* 209.3 (Mar. 2011), pp. 606–625. ISSN: 0890-5401. DOI: [10.1016/j.ic.2010.11.023](https://doi.org/10.1016/j.ic.2010.11.023) (cit. on p. 43).
- [VWW12] Sicco Verwer, Mathijs de Weerd, and Cees Witteveen. “Efficiently Identifying Deterministic Real-Time Automata from Labeled Data.” In: *Machine Learning* 86.3 (Mar. 2012), pp. 295–333. ISSN: 1573-0565. DOI: [10.1007/s10994-011-5265-4](https://doi.org/10.1007/s10994-011-5265-4) (cit. on p. 48).

- [Wal+07] Neil Walkinshaw, Kirill Bogdanov, Mike Holcombe, and Sarah Salahuddin. “Reverse Engineering State Machines by Interactive Grammar Inference.” In: *14th Working Conference on Reverse Engineering (WCRE 2007)*. Ed. by Massimiliano Di Penta and Jonathan I. Maletic. Institute of Electrical and Electronics Engineers, Oct. 2007, pp. 209–218. ISBN: 0-7695-3034-6. DOI: [10.1109/WCRE.2007.45](https://doi.org/10.1109/WCRE.2007.45) (cit. on p. 34).
- [Wal+13] Neil Walkinshaw, Bernard Lambeau, Christophe Damas, Kirill Bogdanov, and Pierre Dupont. “STAMINA: A Competition to Encourage the Development and Assessment of Software Model Inference Techniques.” In: *Empirical Software Engineering* 18.4 (Aug. 2013), pp. 791–824. ISSN: 1573-7616. DOI: [10.1007/s10664-012-9210-3](https://doi.org/10.1007/s10664-012-9210-3) (cit. on p. 37).
- [Wan+20] Fujun Wang, Zining Cao, Lixing Tan, and Hui Zong. “Survey on Learning-Based Formal Methods: Taxonomy, Applications and Possible Future Directions.” In: *IEEE Access* 8 (2020), pp. 108561–108578. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2020.3000907](https://doi.org/10.1109/ACCESS.2020.3000907) (cit. on p. 13).
- [WB08] Neil Walkinshaw and Kirill Bogdanov. “Inferring Finite-State Models with Temporal Constraints.” In: *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. Institute of Electrical and Electronics Engineers, Sept. 2008, pp. 248–257. ISBN: 978-1-4244-2187-9. DOI: [10.1109/ASE.2008.35](https://doi.org/10.1109/ASE.2008.35) (cit. on p. 34).
- [WF04] Priscilla Walmsley and David Fallside. *XML Schema Part 0: Primer Second Edition*. W3C Recommendation. W3C, Oct. 2004. URL: <https://www.w3.org/TR/2004/REC-xmlschema-0-20041028/> (cit. on pp. 90, 91).
- [Wil+16] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C. ’t Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. “The FAIR Guiding Principles for Scientific Data Management and Stewardship.” In: *Scientific Data* 3.1, 160018 (Mar. 2016). ISSN: 2052-4463. DOI: [10.1038/sdata.2016.18](https://doi.org/10.1038/sdata.2016.18) (cit. on p. 91).
- [Win+13] Stephan Windmüller, Johannes Neubauer, Bernhard Steffen, Falk Howar, and Oliver Bauer. “Active Continuous Quality Control.” In: *Proceedings of the 16th International ACM Sigsoft Symposium on Component-Based Software Engineering*. New York, NY, USA: Association for Computing Machinery, June 2013, pp. 111–120. ISBN: 978-1-4503-2122-8. DOI: [10.1145/2465449.2465469](https://doi.org/10.1145/2465449.2465469) (cit. on p. 34).
- [Win99] Russell S. Winer. “Experimentation in the 21st Century: The Importance of External Validity.” In: *Journal of the Academy of Marketing Science* 27.3 (June 1999), pp. 349–358. ISSN: 1552-7824. DOI: [10.1177/0092070399273005](https://doi.org/10.1177/0092070399273005) (cit. on p. 3).
- [Wiś+23] Remigiusz Wiśniewski, Grzegorz Bazydło, Marcin Wojnakowski, and Mateusz Popławski. “Hippo-CPS: A Tool for Verification and Analysis of Petri Net-Based Cyber-Physical

- Systems.” In: *Application and Theory of Petri Nets and Concurrency*. Ed. by Luis Gomes and Robert Lorenz. Lecture Notes in Computer Science 13929. Cham: Springer Nature Switzerland, 2023, pp. 191–204. ISBN: 978-3-031-33620-1. DOI: [10.1007/978-3-031-33620-1_10](https://doi.org/10.1007/978-3-031-33620-1_10) (cit. on p. 38).
- [WKK17] Fabian Witter, Timo Klerx, and Artus Krohn-Grimberghe. “An Interval Distribution Analysis for RTI+.” In: *Proceedings of the 6th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2017)*. Ed. by Maria De Marsico, Gabriella Sanniti di Baja, and Ana Fred. SciTePress, 2017, pp. 351–358. ISBN: 978-989-758-222-6. DOI: [10.5220/0006146603510358](https://doi.org/10.5220/0006146603510358) (cit. on p. 43).
- [Wri+22] Austin Wright, Henry Andrews, Ben Hutton, and Greg Dennis. *JSON Schema Draft 2020-12*. JSON Schema, June 16, 2022. URL: <https://json-schema.org/draft/2020-12> (cit. on p. 91).
- [WTD16] Neil Walkinshaw, Ramsay Taylor, and John Derrick. “Inferring Extended Finite State Machine Models from Software Executions.” In: *Empirical Software Engineering* 21.3 (June 2016), pp. 811–853. ISSN: 1573-7616. DOI: [10.1007/s10664-015-9367-7](https://doi.org/10.1007/s10664-015-9367-7) (cit. on p. 43).
- [WWT02] Wei Wei, Bing Wang, and Don Towsley. “Continuous-Time Hidden Markov Models for Network Performance Evaluation.” In: *Performance Evaluation* 49.1 (Sept. 2002), pp. 129–146. ISSN: 0166-5316. DOI: [10.1016/S0166-5316\(02\)00122-0](https://doi.org/10.1016/S0166-5316(02)00122-0) (cit. on p. 43).
- [XAZ22] Runqing Xu, Jie An, and Bohua Zhan. “Active Learning of One-Clock Timed Automata Using Constraint Solving.” In: *Automated Technology for Verification and Analysis*. Ed. by Ahmed Bouajjani, Lukáš Holík, and Zhilin Wu. Lecture Notes in Computer Science 13505. Cham: Springer International Publishing, 2022, pp. 249–265. ISBN: 978-3-031-19992-9. DOI: [10.1007/978-3-031-19992-9_16](https://doi.org/10.1007/978-3-031-19992-9_16) (cit. on pp. 44, 48).
- [Xu+09] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. “Detecting Large-Scale System Problems by Mining Console Logs.” In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. New York, NY, USA: Association for Computing Machinery, Oct. 2009, pp. 117–132. ISBN: 978-1-60558-752-3. DOI: [10.1145/1629575.1629587](https://doi.org/10.1145/1629575.1629587) (cit. on pp. 37, 38).
- [YAM21] YAML Language Development Team. *YAML Ain’t Markup Language (YAML™) Version 1.2*. Revision 1.2.2. Oct. 2021. URL: <https://yaml.org/spec/1.2.2/> (cit. on p. 92).
- [Yan+19] Nan Yang, Kousar Aslam, Ramon Schiffelers, Leonard Lensink, Dennis Hendriks, Loek Cleophas, and Alexander Serebrenik. “Improving Model Inference in Industry by Combining Active and Passive Learning.” In: *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Ed. by Xinyu Wang, David Lo, and Emad Shihab. Institute of Electrical and Electronics Engineers, Feb. 2019, pp. 253–263. ISBN: 978-1-7281-0591-8. DOI: [10.1109/SANER.2019.8668007](https://doi.org/10.1109/SANER.2019.8668007) (cit. on p. 34).
- [Yan91] Saeyang Yang. *Logic Synthesis and Optimization Benchmarks User Guide. Version 3.0*. Jan. 15, 1991 (cit. on pp. 37, 38, 41).
- [Zak+19] Ilya Zakirzyanov, Antonio Morgado, Alexey Ignatiev, Vladimir Ulyantsev, and Joao Marques-Silva. “Efficient Symmetry Breaking for SAT-Based Minimum DFA Inference.” In: *Language and Automata Theory and Applications*. Ed. by Carlos Martín-Vide, Alexander Okhotin, and Dana Shapira. Lecture Notes in Computer Science 11417. Cham:

- Springer International Publishing, 2019, pp. 159–173. ISBN: 978-3-030-13435-8. DOI: [10.1007/978-3-030-13435-8_12](https://doi.org/10.1007/978-3-030-13435-8_12) (cit. on p. 39).
- [Zha+24] Jianfeng Zhan, Lei Wang, Wanling Gao, Hongxiao Li, Chenxi Wang, Yunyou Huang, Yatao Li, Zhengxin Yang, Guoxin Kang, Chunjie Luo, Hainan Ye, Shaopeng Dai, and Zhifei Zhang. “Evaluatology: The Science and Engineering of Evaluation.” In: *BenchCouncil Transactions on Benchmarks, Standards and Evaluations* 4.1, 100162 (2024). ISSN: 2772-4859. DOI: [10.1016/j.tbench.2024.100162](https://doi.org/10.1016/j.tbench.2024.100162) (cit. on p. 12).
- [Zha21] Jianfeng Zhan. “Call for Establishing Benchmark Science and Engineering.” In: *Bench-Council Transactions on Benchmarks, Standards and Evaluations* 1.1, 100012 (2021). ISSN: 2772-4859. DOI: [10.1016/j.tbench.2021.100012](https://doi.org/10.1016/j.tbench.2021.100012) (cit. on p. 12).
- [Zhu+23] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R. Lyu. “Loghub: A Large Collection of System Log Datasets for AI-Driven Log Analytics.” In: *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. Institute of Electrical and Electronics Engineers, Oct. 2023, pp. 355–366. ISBN: 979-8-3503-1594-3. DOI: [10.1109/ISSRE59848.2023.00071](https://doi.org/10.1109/ISSRE59848.2023.00071) (cit. on pp. 67, 73, 93).
- [ZR08] Lynette van Zijl and Lesley Raitt. “Random Generation of Unary Finite Automata Over the Domain of the Regular Languages.” In: *South African Computer Journal* 41 (Dec. 2008), pp. 57–63. ISSN: 1015-7999. DOI: [10.10520/EJC28060](https://doi.org/10.10520/EJC28060) (cit. on p. 14).
- [ZSG90] Victor Zue, Stephanie Seneff, and James Glass. “Speech Database Development at MIT: Timit and Beyond.” In: *Speech Communication* 9.4 (Aug. 1990), pp. 351–356. ISSN: 0167-6393. DOI: [10.1016/0167-6393\(90\)90010-7](https://doi.org/10.1016/0167-6393(90)90010-7) (cit. on p. 37).
- [ZSU18] Ilya Zakirzyanov, Anatoly Shalyto, and Vladimir Ulyantsev. “Finding All Minimum-Size DFA Consistent with Given Examples: SAT-Based Approach.” In: *Software Engineering and Formal Methods*. Ed. by Antonio Cerone and Marco Roveri. Lecture Notes in Computer Science 10729. Cham: Springer International Publishing, 2018, pp. 117–131. ISBN: 978-3-319-74781-1. DOI: [10.1007/978-3-319-74781-1_9](https://doi.org/10.1007/978-3-319-74781-1_9) (cit. on p. 39).
- [Zub99] W. M. Zuberek. “Petri Net Models of Process Synchronization Mechanisms.” In: *IEEE SMC’99 Conference. 1999 IEEE International Conference on Systems, Man, and Cybernetics*. Vol. 1. Institute of Electrical and Electronics Engineers, Oct. 1999, pp. 841–847. ISBN: 978-0-7803-5731-0. DOI: [10.1109/ICSMC.1999.814201](https://doi.org/10.1109/ICSMC.1999.814201) (cit. on p. 38).