International Journal on Software Tools for Technology Transfer
https://doi.org/10.1007/s10009-026-00839-z

EXPLANATION PARADIGMS LEVERAGING ANALYTIC INTUITION

Regular

A research agenda for active automata learning

Sophie Fortz[1] · Fatemeh Ghassemi[2] · Léo Henry[3] · Falk Howar[4] · Thomas Neele[5] · Jurriaan Rot[6] · Marnix Suilen[7]

Accepted: 3 February 2026

Abstract

We develop a research agenda for the field of automata learning. Automata learning algorithms infer state-machines from observations. The study of such algorithms began in the 1970s and until today has led to a wide range of different learning models, learnability results, and learning algorithms for many different classes of automata as well as to many different applications of automata learning, e.g., specification generation, learning-based testing, and black-box verification. As the field still stratifies and learning algorithms and new applications are conceived, it will be helpful to consolidate and integrate individual obtained results into a coherent set of principles of automata learning and techniques for devising learning algorithms. We aim to provide a step in this direction by conducting a survey of active automata learning methods, focusing on different application scenarios (application domains, environments, and desirable guarantees) and the overarching challenges that emerge from these. We identify concrete research questions through a (short) bibliographic study highlighting the state of the art and the technical implications that are derived from the overarching challenges.

Keywords Automata learning

1 Introduction

Automata are a fundamental concept in computer science, serving as essential tools for, e.g., modelling languages, systems, computing devices, protocols, and games. They are also crucial for studying a broad range of properties such as expressivity, the existence of canonical models, and decidability across various automata classes. Due to their fundamental role and widespread application, automata provide a robust basis for developing algorithms. Among these, automata learning algorithms stand out, aiming to automate the construction of state-machine models based on observed behaviour. First introduced in the 1970s [76], these algorithms have since evolved and found significant practical applications in diverse fields.

Active automata learning is an automated technique for inferring automata models through interaction with a system. In contrast to many other (passive) automata learning techniques that fall within the paradigm of machine learning, active model learning does not aim at extracting patterns from large amounts of data but instead uses the ability to conduct specific tests to infer concise and, in many cases, precise models with guaranteed properties on the inferred models and the learning process, e.g., minimality of models

Marnix Suilen's work primarily done while at Radboud University.

✉ S. Fortz
sophie.fortz@kcl.ac.uk

F. Ghassemi
fghassemi@ut.ac.ir

L. Henry
leo.henry@ucl.ac.uk

F. Howar
falk.howar@tu-dortmund.de

T. Neele
t.s.neele@tue.nl

J. Rot
jurriaan.rot@ru.nl

M. Suilen
marnix.suilen@uantwerpen.be

[1] King's College London, London, UK

[2] University of Tehran, Tehran, Iran

[3] University College London, London, UK

[4] TU Dortmund, Dortmund, Germany

[5] Eindhoven University of Technology, Eindhoven, The Netherlands

[6] Radboud University, Nijmegen, The Netherlands

[7] University of Antwerp, Antwerp, Belgium

Published online: 19 February 2026

and guaranteed convergence during learning. The provided guarantees on models and learning algorithms make active automata learning a valuable technique that can complement model-based techniques like model checking, model-based testing, conformance testing, runtime monitoring and synthesis to enable the application of said techniques without the prerequisite of providing correct models.

Consequently, active automata learning is an active and growing field of research and advances span from theoretical contributions like extensions to new classes of models and efficient algorithms, to integration with other techniques, e.g., refinement loops between learning and verification, to applications of active automata learning in industrial case studies. The most notable (but by far not the only) applications are in the domain of software engineering.

Over the decades, a structure of the field has emerged that makes it possible to characterise contributions and identify types of contributions, types of arguments, and appropriate study designs. One such defining structure is the so-called MAT model of a *minimally adequate teacher*, introduced by Dana Angluin in 1987 along with the L^* learning algorithm [18]. The MAT model to this day is the basis for expressing learning algorithms as a game between a learner, who infers a model, and a teacher, who answers certain types of queries about the *system under learning* (SUL). Similarly, the L^* algorithm still provides an algorithmic basis for extensions of active automata learning to new classes of models.

The L^* algorithm (and many variants thereof) relies on an essential property of classes of models for which active automata learning algorithms can be developed: two words w_1 and w_2 in a set of sequences for the respective class are equivalent iff the observable behaviour after w_1 is equivalent to the observable behaviour after w_2 — for appropriate notions of *observable behaviour* and *equivalence*. For regular languages and finite state acceptors, the Myhill-Nerode relation expresses this property [135].

After forty years of research, active automata learning (or active learning for short) is a mature research field that has a large body of theoretical and empirical works that have led to numerous learning algorithms for varied classes of automata and many practical applications. Yet, there is no shortage of open challenges that have to be overcome to further advance the field, develop new techniques, and ensure industrial impact of the developed techniques. This stratification makes it worthwhile to try and identify potential for consolidation and integration, and to identify overarching themes and research challenges.

This paper presents a short overview of the core active learning bibliography and formulates an organised research agenda. The main goal is to map out interesting, relevant, and attainable research challenges for the field and analyse the progress that has been made towards these challenges as well as to differentiate current technical restrictions and fundamental limitations. We propose four general challenges for the entire field that can be concretised into more specific challenges in individual works or lines of work. We introduce the challenges at a very high level in the following paragraphs before turning to a more detailed analysis of the field. In total, we provide 35 challenges.

Related works There are several surveys dedicated to the current state of active learning research and the theoretical and practical challenges of its application. A 2011 survey [91] identified the challenges for the practical application of automata learning as efficiency, the expressivity of models, the semantic gap between formal models and learned systems, incompleteness of active learning in black box scenarios (defining an adequate interface for membership queries, adequate abstraction, independent membership queries, and implementing the equivalence queries). The next survey [88] exhaustively reviewed the progresses that had been made over the five years from 2011 to 2016 in the application, tools/libraries, and algorithms of active learning. Algorithmic advances in this period include different strategies for analysing counterexamples (cf, e.g., Irfan et al. [93]) and recombinations of earlier algorithms and data structures with the goal of optimising efficiency. One notable result of this period is the TTT algorithm [96]. Another survey in 2017 [170] introduced the recent advances and forecasted huge potential for applications, especially in the area of legacy control software. Handling predicates and operations on data, models more expressive than Mealy machines, evaluating the quality of learned models, and combining the black-box scenario with the white-box techniques are the major challenges discussed in this survey. Finally, a 2021 survey [12] gives a general overview of the field and identifies challenges in the areas of abstraction, equivalence testing, non-determinism and expressivity. All recent surveys agree on immaturity of algorithms and tools to make active learning applicable to a large class of systems.

In this research agenda we elaborate on the challenges with more details and discuss recent progresses towards these challenges.

Motivating example: software verification To start with a concrete application domain that is well within the scope of this journal: software is a part of every system we use and correctness of software is critical in this digital era. There are many approaches to verify and validate the correctness of systems. Using formal methods is the envisioned technique for verification of safety-critical systems [24]. Formal methods use mathematical models for analysis and verification at any stage of the program life-cycle [186]. On top of foundational works, models are directly used in practice in verification and other formal methods (e.g., model-based

testing [35], model checking [42], monitoring [27, 110]), program analysis and simulation or even to generate code [122].

All of these methods have a rich history of industrial applications and their use is becoming more widespread under the pressure of increasingly complex, large, and connected systems. The list of such success stories is too long to be listed in this paper, but some examples include:

- TGV, a tool allowing to generate test cases for non-deterministic reactive systems [98];
- UPPAAL [167], the main model checking and testing tool for timed systems (based on timed automata), that led to numerous case studies and papers.
- PRISM [108] and Storm [85], tools for probabilistic model checking.

In some domains, formal methods are now recommended in the design, development and verification of systems [24], to further the quality of these systems in practice [86].

However, all of the above mentioned methods depend on the existence of models faithfully representing a system's properties of interest with a fitting degree of abstraction. Models should be sufficiently abstract to be manageable by algorithms (and potentially humans) while being sufficiently precise to capture the relevant behaviours of the system with respect to the properties of interest. Yet, in most of the industry, models are rarely produced along with the systems, meaning they need to be created *afterwards*, for the purpose of the above-mentioned methods.

Hand-crafting models is slow, error-prone, and requires expertise in both the system domain and the model class used. It is often too much of an entry cost for formal methods, limiting their applications. Furthermore, legacy systems exist that have no clear specification or model and yet play crucial roles in numerous software environments. Ensuring that their replacements retain their desirable properties is thus often impossible. There are techniques based on program analysis that construct models from source code, but they are often limited by the presence of library modules, third-party components, etc., that make analysis of source code difficult. One answer to this issue is to automate the model creation process, which is the objective of *active automata learning*.[1]

Challenges From the example above, we can derive several requirements or challenges for active automata learning algorithms (or for any model-generation technique for that matter):

1. Models must reflect the behaviour of a system symbolically and abstractly, yet faithfully to obtain meaningful guarantees.
2. Models must capture behavioural properties of a system at a level of abstraction that balances usefulness and efficiency / computability of guarantees.
3. The model generation approach should be adapted based on the degree of access to system internals, ranging from running legacy applications to having full access to source code.
4. Models must be produced in an automated way from existing systems or design artefacts.

These requirements extend beyond the application of model learning in the context of software verification. They pertain to any scenario in which inferred or generated models must be an accurate representation of a target for some purpose (e.g., protocol conformance testing, controller synthesis, or model checking). To meet these requirements, or to decide that they cannot be met in some intended application scenarios, it will be important to find general answers to the following questions.

Challenge 1
Which types of models can be learned?

The type of models that can be learned is impacted by their expressive power. Active automata learning algorithms usually are based on a (Myhill-Nerode type) congruence on the languages corresponding to an automaton class. While more expressive models can represent complex behaviours more precisely, there may not exist corresponding languages with congruence relations. Even with congruence relations, learning may quickly become very expensive when data values or large state-spaces are involved. It is not yet clear where the limits in expressiveness and learnability are.

Challenge 2
What properties make models useful?

A model's utility depends on its interpretability or utility in further analyses. While many applications have been explored, it has not yet been comprehensively studied, which properties of learned models are important (e.g., completeness, faithfulness to the learned system, size of the model, captured phenomena, etc.).

Challenge 3
In which contexts can we apply active learning algorithms?

While active automata learning is often phrased in the MAT model in which a teacher answers membership and equivalence queries, this model may not be appropriate in all contexts or does not completely characterise learning scenarios. In some cases, e.g., automata learning may be combined

[1] We choose this term rather than active learning or active model learning to better pinpoint the community at play.

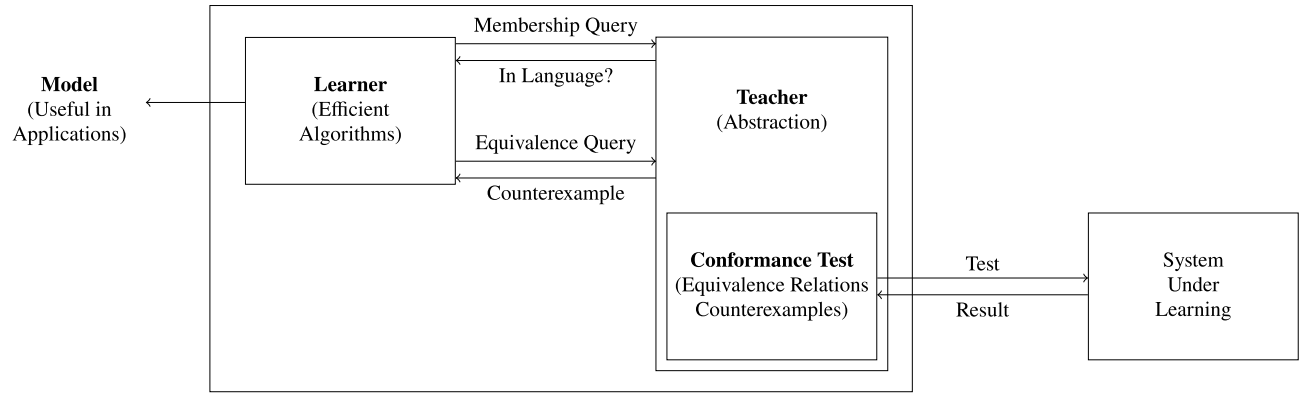


Fig. 1 Schematic view of active automata learning in the MAT model as a game between learner and teacher. The teacher answers the learner's questions about the behaviour of the system under learning. The learner aggregates these answers into a behavioural model. The teacher provides an abstraction that yields useful models and uses conformance testing methods to validate or disprove candidate models conjectured by the learner.

with other methods for inferring control-structure and data-relations. In other scenarios, additional access to recorded traces may exist or resetting a SUL into a default starting state between learning episodes may not be possible.

Challenge 4
What actions need to be taken to be able to employ active learning?

Active automata learning requires some infrastructure to infer models of actual systems. And actual systems may not immediately be amenable to automata learning as they, e.g., run in noisy and complex environments and can only be accessed through other components. We currently lack a comprehensive overview of techniques for making systems amenable to learning.

Outline We organise the remainder of the paper as follows. We begin with a gentle introduction to automata learning, highlighting key success stories in Sect. 2. Section 3 explores the general characteristics of qualitative models, while specific application contexts are highlighted in Sect. 4. In Sect. 5, we discuss the implementation of active learning and the associated challenges. Section 6 provides an in-depth technical review, detailing the state of the art and central technical challenges. Finally, we conclude this agenda in Sect. 7.

2 A brief overview of active automata learning

This section gives a broad overview of the field of model learning in terms that can be understood by a generalist audience and give a gentle introduction to the basic algorithmic concepts. The section then provides some examples of successful applications, e.g., to aid verification of systems or as a means of explaining system behaviour to users.

2.1 A gentle introduction to algorithms

Automata learning algorithms are an important technique for understanding and modelling the behaviour of systems that exhibit finite state or sequential processes. The behaviour of the system under learning (SUL) can be conceptualised as an automaton (or sometimes a language), with its input and output alphabets serving as its vocabulary to communicate within its environment. The input and output alphabets describe the actions or data that the SUL can accept as input or produce as output, respectively.

According to the MAT framework, active automata learning revolves around the interaction between a teacher and a learner, as shown in Fig. 1. The learner acquires knowledge about the SUL through the iterative process of observation, hypothesis generation, and refinement. The teacher, which is assumed to have knowledge about the SUL, interacts with the learner by responding to queries and providing new information about the SUL's behaviour. Throughout the learning process, the learner generates hypotheses or conjectures about the system's behaviour and refines them iteratively based on feedback from the teacher. Hypotheses generated by the learning algorithm are tentative models that approximate the behaviour of the target system based on the observed data. The ultimate goal is to converge towards a hypothesis that accurately represents the SUL. This final hypothesis allows for effective modelling and analysis of complex systems.

Central to the process are *membership* and *equivalence queries*. Membership queries involve providing inputs to a system and observing its corresponding outputs, allowing learners to determine whether a given input-output pair is

allowed by the system. These queries help the learner gradually build an understanding of the system's dynamics in the form of a hypothesis automaton. Equivalence queries, on the other hand, assess whether the inferred model behaves equivalently to the target system across all possible inputs. The teacher responds with a counterexample if the hypothesis fails to capture the system accurately, guiding the refinement of the learner's model. Otherwise, the hypothesis is validated, and the learning terminates.

Counterexamples play a pivotal role in the iterative refinement of hypotheses generated by the learning algorithm. They represent instances where the inferred model diverges from the behaviour of the target system, prompting adjustments to the hypothesis. Positive and negative examples are instances provided by the teacher to illustrate the desired behaviour of the system (positive) or highlight behaviours to be avoided (negative). The implementation of this framework by the L^* algorithm is detailed in Explanation 1. Explanation 2 presents a running example of the L^* algorithm on a simple communication protocol.

A determining factor in the performance of the learning process is the data structure in which observations are stored. In the case of L^*, this is an *observation table*. However, the way L^* integrates counter-examples into the table may introduce redundant rows and columns that represent identical states (requiring more membership queries to fill all the cells). This is partially resolved by the approaches of Maler and Pnueli [114], who extend the columns instead of the rows, and Rivest and Schapire [147], who add just one row for each counter-example. A fundamentally different approach is the Kearns-Vazirani algorithm [101], which replaces the observation table with a discrimination tree in which only distinct states are maintained, leading to a smaller number of membership queries. The leaves of an observation tree correspond to the prefixes (a word that reaches a given state) and the inner nodes are distinguishing suffixes. Each two pair of prefixes is discriminated with their lowest common ancestor in the tree. TTT [96] further improves on this idea by compacting the required information into three tree-like data structures: a spanning tree defining unique prefixes, embedded into the hypothesis' transition graph, a discrimination tree, and a discriminator tree for storing the suffix closed set of discriminators. $L^\sharp$ [171] follows a new perspective based on the concept of apartness, a constructive form of inequality. It is conceptually simpler than TTT (while achieving the same asymptotic complexity) since it does not rely on any auxiliary data structure and instead operates directly on an observation tree from which the hypothesis is ultimately constructed. Furthermore, it exploits adaptive membership queries, choosing the next input dynamically based on the output observed so far, to discriminate prefixes with a smaller number of membership queries. Finally, L^λ [89] is the first algorithm that avoids directly integrating counter-examples in its data structure but instead uses them to guide a type of lazy partition refinement.

Explanation 1: L^* Learning Procedure Step-by-Step

We explain how the MAT framework is implemented by Dana Angluin's L^* algorithm [18] for learning a *deterministic finite automaton* (DFA). Given an alphabet Σ, a DFA is a tuple $\mathcal{A} = (Q, \delta, q_0, F)$ where Q is a set of states, $\delta\colon Q \times \Sigma \to Q$ is a transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is a set of *final* or *accepting* states. We lift the transition function to *words*, such that $\delta(q, \epsilon) = q$ (where ϵ is the empty word) and $\delta(q, w \cdot a) = \delta(\delta(q, w), a)$ for all $q \in Q$, $w \in \Sigma^*$ and $a \in \Sigma$. The *language* of $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$. We remark that the ideas below can easily be generalised to Mealy machines, by the fact that a DFA can be seen as a Mealy machine with $\{0, 1\}$ as the set of output symbols.

L^* maintains a so-called *observation table* (S, E, T), where $S \subseteq \Sigma^*$ is a prefix-closed set of words that label the rows, $E \subseteq \Sigma^*$ is a suffix-closed set of words that label the columns and $T\colon (S \cup S \cdot \Sigma) \times E \to \{0, 1\}$ fills the cells of the table. We also use the representation $row\colon S \cup S \cdot \Sigma \to E \to \{0, 1\}$ such that $row(w)(u) = T(w, u)$ for all $w \in S \cup S \cdot \Sigma$ and $u \in E$. Initially, we set $S = E = \{\epsilon\}$. To fill the table, the learner poses membership queries $w \in \mathcal{L}(\mathcal{A})$ for some word w and updates T accordingly. While doing this, two crucial properties are maintained, namely *closedness* and *consistency*. A table (S, E, T) is closed iff for all $w \in (S \cdot \Sigma) \setminus S$, there is a $w' \in S$ such that $row(w) = row(w')$; a table can be closed by extending S with violating words. A table is consistent iff for all $w, w' \in S$ such that $row(w) = row(w')$, it holds that $row(w \cdot a) = row(w' \cdot a)$ for all $a \in \Sigma$; an inconsistency $row(w \cdot a)(u) \neq row(w' \cdot a)(u)$ is resolved by adding $a \cdot u$ to E. Meanwhile, T is extended by membership queries.

Once this process terminates, L^* generates a hypothesis from its closed and consistent observation table, namely the DFA $\mathcal{H}(S, E, T) = (Q, \delta, row(\epsilon), F)$, where $Q = \{row(w) \mid w \in S\}$, $\delta(row(w), a) = row(w \cdot a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$ and $F = \{row(w) \mid w \in S, row(w)(\epsilon) = 1\}$. This DFA serves as a hypothesis of the target system. If the teacher rejects this hypothesis with a counter-example w, then w and all its prefixes are added to S (and the process repeats). Otherwise, the learning terminates successfully with the result $\mathcal{H}(S, E, T)$.

Implementations On the practical side, various frameworks for automata learning are readily accessible online, offering tools and implementations to support the learning process. Among these, LearnLib [97, 123, 146] stands out as a prominent example, providing a Java-based implementation of the L^* algorithm and various extensions. LearnLib

Explanation 2: Running example of the L^* Algorithm

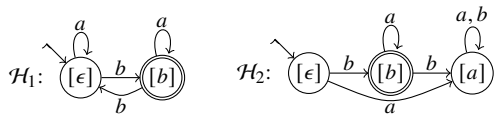
Consider a simple communication protocol with an initialization phase (b), followed by sending messages (a), specified by ba^*. We apply the L^* algorithm to learn this protocol. The initial table $\mathcal{O}_1$ is not closed as $row(b) \neq row(\epsilon)$. After adding b to S, the table $\mathcal{O}_2$ is closed and consistent and the hypothesis $\mathcal{H}_1$ is generated which results in ab as an counterexample. By adding ab and its prefix a to S, the table $\mathcal{O}_3$ is not consistent as $row(\epsilon \cdot b) \neq row(a \cdot b)$, leading to expanding the table by column b. The table $\mathcal{O}_4$ is closed and consistent and the algorithm terminates with $\mathcal{H}_2$.

$\mathcal{O}_1$	ϵ
ϵ	0
a	0
$\boldsymbol{b}$	**1**

$\mathcal{O}_2$	ϵ
ϵ	0
b	1
a	0
ba	1
bb	0

$\mathcal{O}_3$	ϵ
$\boldsymbol{\epsilon}$	**0**
b	1
$\boldsymbol{a}$	**0**
ab	0
ba	1
bb	0
aa	0
aba	0
abb	0

$\mathcal{O}_4$	ϵ	b
ϵ	0	1
b	1	0
a	0	0
ab	0	0
ba	1	0
bb	0	0
aa	0	0
aba	0	0
abb	0	0



offers a comprehensive suite of functionalities for automata learning, including support for active and passive learning modes, several equivalence approximation strategies, and support for different formalisms. AALpy [130] is another notable framework offering a Python-based environment for automata learning. While LearnLib and AALpy offer robust solutions for automata learning, it is worth noting that `libalf` [32], although no longer actively maintained, also provided valuable resources for automata learning implementations. These frameworks serve as valuable resources for researchers and practitioners alike, enabling the application of automata learning techniques to a wide range of real-world problems. For a detailed comparison of the features of each tool, see Table 1.

Other frameworks Another category of algorithms is *passive learning*. In active learning, the learner actively selects which examples to query the teacher for, while in passive learning, the learner solely relies on the information provided by the teacher [38, 76, 138]. As a consequence of this, passive learning algorithms typically provide fewer guarantees about the quality of the resulting model. An example of such an algorithm is *RPNI* (Regular Positive and Negative Inference) [48].

A line of work that is closely related to MAT-style active learning is based on interactive state-merging, integrating passive learning algorithms and model-based testing techniques [6]: the original QSM framework [54] alternates between system sampling and passive inference over the col-

Table 1 Comparison of automata learning libraries

		LearnLib[b]	AALpy[c]	libalf[d]
	References	[123, 146]	[130]	[32]
	Language	Java	Python	C++, Java
	Still Maintained[a]	✓	✓	✗
Supported formalisms	DFA	✓	✓	✓
	Markov Chains	✗	✓	✗
	Markov Decision Processes	✗	✓	✗
	Mealy Machines	✓	✓	✓
	Moore Machines	✓	✓	✗
	NFA	✓	✗	✓
	Non Deterministic Moore Machine	✗	✓	✗
	SBA	✓	✗	✗
	SPA	✓	✗	✗
	SPMM	✓	✗	✗
	SST	✓	✗	✗
	Stochastic Mealy Machines	✗	✓	✗
	(Abstracted) ONFSM	✗	✓	✗
	Visibly 1-counter	✗	✗	✓
	VP(D)A / SEVPA	✓	✓	✗
Active algorithms	AAAR	✓	✗	✗
	ADT	✓	✗	✗
	DHC	✓	✗	✗
	Kearns & Vazirani	✓	✓	✓
	Kearns & Vazirani$_{\text{VPA}}$	✗	✓	✗
	λ (L^λ, TTT^λ)	✓	✗	✗
	$L^\#$	✓	✓	✗
	L^*	✓	✓	✓
	NL^*	✓	✗	✓
	Observation Pack	✓	✗	✗
	(Abstracted) ONFSM L^* / $L^\#$	✗	✓	✗
	Procedural	✓	✗	✗
	Stochastic L^*	✗	✓	✗
	TTT	✓	✗	✗
	TTT$_{\text{VPA}}$ / Observation Pack$_{\text{VPA}}$	✓	✗	✗
	Visibly 1-counter automata	✗	✗	✓
Passive algorithms	ALERGIA	✗	✓	✗
	Biermann	✗	✗	✓
	DeLeTe2	✗	✗	✓
	EDSM	✗	✓	✗
	k-tails	✗	✓	✗
	OSTIA	✓	✗	✗
	PAPNI	✗	✓	✗
	RPNI	✓	✓	✓

Table 1 (*Continued*)

		LearnLib[b]	AALpy[c]	libalf[d]
Equivalence approximation strategies	Black-Box checking	✓	✗	✗
	Breadth-first search	✓	✓	✗
	k-Way state/transition coverage	✗	✓	✗
	PAC	✗	✓	✗
	Perfect knowledge (white-box)	✓	✓	✗
	Random walk	✓	✓	✗
	Random words	✓	✓	✗
	State prefix	✗	✓	✗
	Transition focus	✗	✓	✗
	User-provided input sequences	✓	✓	✓
	W-method	✓	✓	✗
	Wp-method	✓	✓	✗
Total number of equivalence oracles		14	13	1[e]
Other features	Abstraction	✓	✓	✗
	Caching	✓	✓	✗
	Extensible	✓	✓	✗
	Filters	✓	✗	✓
	Imports / Exports	✓	✓	✗
	Normalisers	✗	✗	✓
	Parallelism	✓	✗	✗
	Statistics	✓	✓	✓
	SUL Adapter	✓	✓	✗
	Visualization	✓	✓	✓

[a]Information in this table was retrieved as of April 3, 2025. Any updates, new functionalities, or developments after this date are not reflected.

[b]Main page: https://learnlib.de/. Github: https://github.com/Learnlib/learnlib.

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

[d]Github: https://github.com/libalf/libalf.

[e]libalf documentation does not specify any equivalence query strategies. However, the source code suggests that it primarily relies on user-provided counterexamples.

lected traces. Walkinshaw *et al.* [70, 183] improve upon the QSM algorithm and demonstrate how a learned hypothesis can guide further queries and how users can input knowledge about incorrect transitions in hypothesis models to the learning algorithm.

More recently, Vaandrager and coauthors' $L^{\#}$ algorithm [171] can be seen as a direct continuation of the integration of ideas from passive learning and learning algorithms in the MAT framework: it maintains an ever-growing prefix-tree acceptor of observed behaviours and dynamically computes apartness (the dual of equivalence) between states during query generation. This, as one could argue, emulates state merges on the fly rather than in a separate batch phase. This approach seems to hold a lot of potential for future development. The tight integration of active querying with passive, merge-driven restructuring of the hypothesis blurs the boundary between classical state merging and discrimination, yielding scalable and robust learners that have the potential to deal gracefully with non-determinism in observations.

PAC (Probably Approximately Correct) learning [172], a concept borrowed from statistical learning theory, provides a theoretical framework for quantifying the accuracy and efficiency of learning algorithms. It ensures that the learned model is likely to be correct with high probability based on the observed data. This framework allows for a probabilistic analysis of the learning algorithm's performance and generalisation capabilities.

2.2 Successful applications

Automata learning has seen a wide array of successful applications to extract the behaviour of various real-world systems. We categorise the successful applications based on the categories established in an earlier structured literature survey [88] and expand on this by also categorising works that have appeared after the aforementioned survey, i.e., since 2016.

Specification generation The most obvious application of automata learning is the generation of a posterior specification from the running systems after their implementations. Several studies show how to apply learning to test an implementation of a communication protocol, e.g., TCP [63, 64], TLS [49], SSH [65], Bluetooth LE [143], QUIC [60], MQTT [161] and IKEv1 [144]. Taking the example of TCP, one can interact with the implementation by sending it packets that follow the format set in the TCP standard [55] and recording the responses. The standard also provides hints on which data fields may be abstracted. The resulting state machine may be compared against the state machine that the standard prescribes [55, Fig. 5]. In the case of Fiterau-Brostean et al. [64], the TCP implementation of each operating system they analysed deviated in some way from the standard.

Automata learning may also be applied to extract the behaviour of proprietary hardware, such as bank cards [2], CPU caching policies [180], a two-factor authentication device [41] and bio-metric passports [1]. In some cases, the protocol testing approach explained above can be applied, e.g., bank cards [2] and bio-metric passports [1] each implement a standardised protocol and the communication protocol of the 2FA device in Chalupar et al. [41] was reverse-engineered in earlier work.

Automata learning is successfully applied to extract autonomous driving software modules implemented in MATLAB [153], such as the Lateral State Manager (LSM), a

sub-component of the decision-making and planning module responsible for managing modes during an autonomous lane change [152].

Model-based testing without a prior model The inferred models can be used for test case generation of embedded components [154] and automotive applications [104] to mention some.

Software re-engineering Another application is reverse engineering of legacy software [23, 150], possibly with data [149]. These studies all focus on industrial control software, which naturally contains input/output behaviour to interact with the hardware it controls. The 2019 RERS challenge [99] featured such models, which were obtained through automata learning by the Dutch company ASML. The automata obtained through learning can be applied in refactoring [150] or moving to model-based software engineering.

Assume-guarantee reasoning Automata learning can also be applied in compositional verification to automate part of assume-guarantee reasoning. Here, learning helps to obtain useful abstractions of a single component. This works for probabilistic automata [59], timed systems [112] and model repair [73].

Synthesis Finally, automata learning is used to construct formal models of safe behaviour as a basis for synthesising safe mediators or controllers. Active learning and supervisory control are integrated in Farooqui et al. [57, 58], allowing the application of supervisory control techniques in the absence of *plant models*, that is, models of existing components the supervisor is supposed to interact with. A similar approach based on passive learning allows for *shielding* of reinforcement learning agents [163].

3 Useful models

Symbolic models produced by active automata learning algorithms can provide a basis for formal methods while (in principle) being comprehensible to humans. Their usefulness hinges on actually fulfilling one or both of these purposes. In this section, we derive concrete challenges for two research areas for which formal methods are needed to better manage the risk associated with emerging technologies but models are hard to come by: complex hybrid (autonomous) systems and machine learning.

3.1 Hybrid, complex and large systems

While the theory of active learning is well-developed for DFAs and Mealy machines, that are targeted by all the previously mentioned classical algorithms, the picture is less clear for classes of models describing richer behaviours. In practice, most systems depend on a combination of external factors (data, input) and internal factors (time, probabilities non-determinism, memory) that can easily go beyond what can be modelled by DFAs. It is often the case that some of these factors can not be abstracted from for the modelling task at hand. Formal methods have adapted to numerous classes of expressive models (e.g. symbolic automata, register automata, timed automata, probabilistic automata. . .) and active learning, in turn, is required to provide the necessary models.

Challenge 5
Can we devise learning algorithms for models extending classical automata with the elements those systems depend on?

On top of modelling a specific kind of expressive behaviour, complex systems can require to *combine* different types of expressiveness. A prime example of such models are *cyber-physical systems* (CPSs), i.e., systems that have both physical and software components as a core of their functions (e.g. machine-tools, robots, transportation systems). CPSs can be large (physically and in terms of state-space) and interact with the world in a large array of ways, usually depending on time and other continuous quantities. Modelling such systems is notoriously challenging, mostly due to the hybrid nature of CPSs that requires expertise in several areas, as well as in modelling [50].

A research agenda for cyber-physical systems identifies a few key-points to address [136]. First, there is a need for a general learnable model that can deal with both timed and hybrid constraints. Second, reasoning on the level of individual components is preferred for identifying the reason for certain behaviour; techniques for this are required. This, in turn, shapes some objectives for active learning methods:

- Learn models that combine several extensions of classical automata, including timed and hybrid constraints.
- Learn models of component-based systems that represent the various components and not only their composition.

Challenge 6
Is it possible to infer expressive multi-faced models at the level of components in order to verify at the level of a component?

3.2 Models for explainability

Explainability has become a hot research topic in both artificial intelligence [87] and cyber-physical systems [77]. These systems, especially autonomous systems, often control our daily tasks, based on the context and environmental changes.

Users trust such systems more when they can comprehend the system behaviours and the reasons for each autonomous decision. Thus, to enhance understanding of the system and collaboration between the user and the system, the decision of the system should be explained to the user, i.e., why a certain action was performed [34, 111]. The generation of explanation can be directly encoded into the system code or dynamically performed at runtime based on the purpose and type of the explanation, the recipient, and the level of required details. The latter can be achieved through *explainable models*: "behavioural models that capture the causal relationship among events and system reactions" [30]. Explainable models can be constructed automatically based on (1) executable behaviour models at design time [30, 151], (2) observed behaviours using active automata learning techniques, or (3) the log data [51]. In case an existing formal specification to be used for the extraction of an explainable model does not exist, active learning comes into play. One advantage of explainable models achieved through learning (over ones constructed at design time or derived from logs) is their completeness, meaning that they have extracted from what truly has been implemented and so considered all possible behaviours of the system. Still, we wonder how this works out in practice:

Challenge 7
In what scenario does a user benefit from learning an explainable model?

As the learned model is generated based on user interactions and system outputs, it can explain the causal relation between user inputs, outputs, and the internal state of the system. However, the generated model may be huge and may not carry the right level of abstraction for human understanding. So, there is a need to either direct the active learning process to generate a more abstract automaton that summarises the causal relations adequately or synthesise additional explanatory models like logic formulae [22]. Deciding which input/outputs (and their data parameters) should be considered for explainability while controlling non-determinism due to abstraction is a challenge. One way of controlling the complexity of the final model is generating an abstract symbolic representation of training data before the application of automata learning [51].

Challenge 8
How can model learning techniques provide a comprehensible explanation to users?

Given the relation between artificial neural networks and finite automata [105, 137], the active learning approach has been successfully applied to capture the input-output behaviour of recurrent neural networks [118, 131, 184]. By generating a symbolic oracle from training data, AI functions can be actively learned as timed models (1-timer Mealy machines) [51]. This is an active research field, and we believe the application of active automata learning has significant potential here.

Challenge 9
How can we use model learning techniques for generating symbolic models of ML systems?

4 Application scenarios

In the basic MAT model, it is assumed that the learner can ask queries from the teacher, and the teacher is perfectly capable of providing answers. The learner may ask as many questions as she requires to learn a model. The complexity of learning algorithms depends on the size of the final model. There may not be a fitting model in all practical application contexts. Complete models may be far too large to learn in a limited time and may not even be required for the target task (e.g., verification of specific properties, explainability, etc.) in many contexts. Finally, in practice, a system may change over time, especially when exercised a lot. Active learning algorithms have to be adapted to such conditions.

4.1 Incremental learning

Because CPSs and other complex systems are, by nature, expressive and contain different aspects of expressiveness, they are particularly prone to scalability issues. Indeed, there is a multiplicative interaction among the aspects. For a system having control states, time (expressed through clocks), and memory/data (expressed by registers), the semantic configuration of the system depends on the current control state and the valuation of all clocks and registers. This multiplication makes approaches that limit the cost of each aspect of expressiveness especially desirable. These approaches are often referred to as *lazy*, as they postpone work (computations, system tests, queries, etc.) as much as possible. We can incrementally consider each expressive aspect by employing a lazy approach during the learning process instead of considering all aspects at once.

Challenge 10
Can we develop lazy learning methods for expressive classes of models, to avoid paying the complexity of the class of hypotheses and only pay for the complexity of the SUL?

Challenge 11
Can we develop a learning method that considers aspects incrementally to mitigate the complexity of the multiplication interaction among the aspects?

4.2 Adaptability and concept drifts

Active model learning methods generally suppose that the SUL is a *fixed* object with a unique behaviour corresponding to a language in a given class. However, the reality of systems can be much more nuanced: computer systems can evolve over time (e.g., through software updates); a general piece of software can have options that can be selected during production or deployment, leading to a family of related implementations (e.g., integrated development environments, Linux kernels, etc.). The configuration of a system may dynamically change during its execution to adapt to the changes occurring in the environment.

Evolution through time Adaptability is a feature of learning algorithms that allows them to handle changes in the SUL due to environment changes or software updates. It has first been introduced in the context of model checking [80] as a follow-up to black-box checking, allowing the use of an inaccurate (outdated) model. Many adaptive learners start from pre-seeded information representing the observations gathered during the previous runs, and check them when a new learning phase begins [40, 46, 61, 92, 107, 185]. These need to be informed that a change has occurred to re-start learning, and even if this was automated, change is intrinsically considered as *sparse*: a learning phase should go through without updates to the system in order to dispose of a model representing the system at some point in time. A learning framework allowing the use of any MAT learner in an adaptive setting, named CƐAL, has recently been introduced [62]. This framework allows us to detect changes in the system automatically and to repair the hypothesis on the fly without needing a learning phase to converge without updates. This framework induces an extensive computation cost to reduce the number of system tests.

Spatial variability Aside from time variability, one could also consider spatial variability. *Variability-intensive systems* (VISs) are systems whose characteristics and behaviour can be modified by the activation or deactivation of some options, called features. VISs allows to derive multiple program variants forming a family of related systems.

Software product lines (SPLs) are probably the most famous structured approach to engineer VISs. While a lot of work consider the definition of valid variants, i.e., variants formed of a valid combination of features, there also exists several formalism to represent the behaviour of a VIS (e.g., *featured finite state machines* (FFSMs) [72] and *featured transition systems* (FTSs) [45]).

In [46], the authors defined an adaptation of L^*, called Partial-Dynamic L^* (δL^*_M). δL^*_M updates FFSMs by adding the behaviour of a new product to an existing FFSM. They extend this idea with $FFSM_{DIFF}$ [47], the first fully automated approach to learn a behavioural model of an SPL from scratch. An improved method not only takes previous learned automata into account, but reuses observation tables as well [165]. However, since the number of products is exponential over the number of features, both these approaches [47, 165] are limited to a few sampled variants.

Approaching the variability problem from a different angle, Fortz *et al.* [66] approach learning FTSs by adapting techniques from automata learning. Their approach introduces a novel definition for both the learner and teacher components. Unlike previous methods [47, 165], which required the learner to reconstruct variability from individual product automata, [66] proposes a knowledgeable teacher capable of correlating behaviour with variability aspects. This new perspective eliminates the need for exhaustive enumeration and learning of each product, instead treating variability as a fundamental aspect of the system.

The concept of a more intelligent teacher is compelling, contingent upon the availability of such knowledge. Consequently, there is a need to establish a mapping between system variants and execution logs. VaryMinions [67, 68] employs *recurrent neural networks* to trace software variability within extensive logs. However, logging operations are resource-intensive [37], demanding considerable memory for detailed information. To circumvent excessive logging, existing logs can be analysed to compute and synthesise complex information. In this context, VaryMinions aims to map execution traces, stored in event logs, to variant products of the SPL. Despite the promising outcomes, the approach faces challenges such as the combinatorial explosion of variant numbers, as highlighted by the authors. Hence, Fortz et al. [69] proposed a novel method emphasising a more refined mapping between execution traces and features. However, at this stage, this new approach remains only methodological and necessitates further exploration and refinement.

A tutorial [129], entitled "Automata Learning for Dynamic Software Product Lines", summarises all the current progress in the domain of learning spatial variability.

Challenge 12
Can we learn evolving systems in a more general setting?

Challenge 13
Can we reduce the cost of learning evolving systems?

5 Implementing automata learning in practice

One of the key points of active learning is that the MAT agents - Learner and Teacher - communicate in an abstract alphabet. While this level of abstraction is helpful to produce sufficiently abstract models and design the Learner's algorithm, it is then necessary to translate this alphabet to a more concrete set of actions for the actual system. This has often been done with so-called *mappers* that implement an abstraction. Hand-crafting mappers and system adapters is tedious and limits the applicability of active learning.

More generally, while active learning can be considered a mature field when discussing the actual learning, the discussion is much more muddled with respect to connecting that learning to the world.

This section starts the discussion on the actions to be taken to ensure the applicability of active learning algorithms by discussing the use of interfaces with other fields and the specific challenges of some complex environments.

5.1 Combinations with other fields

One way to alleviate some of the implementation challenges is to rely on other fields to handle them. For example, learning of CPS models has been performed with a lazy approach [7] (see also Sect. 4.1) by relying on model-based testing to discover the hard to reach behaviours of these models and on machine learning (specifically recurrent neural networks) to handle the complex nature of the information, while model learning provides an automaton's structure. This approach has been tested in practice to learn the behaviours of a platoon of vehicles and demonstrated significant improvements when compared to the state of the art approach (based on neural networks without testing and active learning to direct the data-search).

Another example of such a combination occurs in fingerprinting of mobile applications [117]: while observing the sequence of packets emitted, active learning is used to derive a language as a fingerprint by analysing the temporal relationships between packets. Deep learning classifiers are then used on top of this abstraction to return the final result. This combination of approaches allowed to vastly outperform other state of the art techniques with 95% accuracy.

Some connections of active learning with reinforcement learning also exist [132, 160], see the paragraph Unobservability in the next subsection for a more in-depth discussion.

Challenge 14
How can the practical application of automata learning benefit from other techniques?

Finally, although in theory all combinations of extensions of DFAs would be of interest, in practice some may be more prevalent than others. Knowing what properties are most needed would be an interesting compass to focus technical development.

5.2 Complex environments

Independently from the complexity of a SUL, its environment can provide challenges for the learner. For instance, autonomous systems have to handle variations and uncertainties in their environment; software systems are more often than not distributed, and specific components can sometimes only be accessed through other interactions with other components, while the system is online (e.g., network protocols, banking protocols). This can cause noise in the queries or complex behaviours due to the environment such as concurrent systems, unobservable behaviours or the inability to reset the system.

Challenge 15
How can we learn models in complex environments?

Noise Noise is a general term encompassing anything that can affect the nature of the query being sent by the learner, or the answer obtained from the system. Noisy environments mean that some queries to specific answers are *incorrect*. Classical MAT algorithms can not handle this uncertainty, and implementation tricks have been the standard way to circumvent the problem (e.g., repeating queries and making a majority vote). The efficiency of such methods has been studied in the past [145], as well as theoretical limits on the effects of noise [21]. More recently, the effects of noise on L^* [103] have been studied.

In practice however, noise is handled with a multiplication of queries to avoid having a faulty observation reaching the learner, which severely impedes scalability. For instance in [143], noise due to message losses, connection errors and message delays for learning the model of Bluetooth LE handled by repeating queries multiple times and discarding outlier traces. Generally speaking noise introduced by faults such as message loss or other environmental influences lead to non-determinism in the model. Applying approaches that learn either stochastic or non-deterministic automata encode the noise directly into the resulting model. To remedy the non-determinism from the model, [8] masks detected non-determinism with sink states, which then allows to directly employ L^*. CƐAL [62], a framework allowing to handle noise on top of MAT learners, decreasing the number of repetitions required at the cost of computational power has recently been introduced. The framework, although general in its approach, focuses for now on DFAs and Mealy Machines.

Despite these contributions, no general consensus on the handling of noise has emerged, especially for complex systems that are both more prone to scalability issues and less equipped with algorithms.

Challenge 16
How can we better handle noise during the learning phase?

Concurrency In many complex systems, components cannot simply be taken out of their context, making it more challenging to apply model learning to them. Solving this issue can be approached either as treating the whole complex system as a concurrent system, and learning its individual components instead of their product (e.g., [4, 133], see paragraph Compositional techniques in Sect. 6.2 for more), or by applying assume guarantee methods combining verification and learning. This has been done for example for model repair [73] and for timed models (event-recording automata) [112].

Challenge 17
How can we learn components of a complex system that can not be taken apart? Can we avoid the exponential cost of the product?

Unobservability Complex systems can be made of multiple communicating components or have internal states that change without inputs or outputs. From a formal point of view, this corresponds to *unobservable actions or transitions* in the model. This problem is well known in the model checking community and has given rise to black-box checking [140], which combines model-checking and testing through games with imperfect information. In the active learning community, however, this problem is scarcely studied—see [100] for a work concerning *partially observable Markov decision processes* (POMDPs) [157]. While the problem is clearly hard for learning, as one wants to infer a model *from observations*, some work-around may exist—possibly for restricted cases.

POMDPs are well studied in *reinforcement learning* (RL), where one wants to learn a controller for a stochastic environment modelled as a POMDP. Although the learning problem is generally intractable (synthesising a controlling for a *known* POMDP is undecidable in many cases [113]), some results exist for restricted cases.

The general approach to dealing with partial observability in RL is to include some form of memory to keep track of previous interactions in an attempt to distinguish separate states. Such approaches, for instance, assume the underlying POMDP is actually k-Markovian [33, 155], include an adaptive memory structure that is updated in the presence of new information [119–121], or apply abstraction methods [158]. For a complete overview of the state-of-the-art in (model-based) RL, we refer to surveys [124, 128].

In [132], RL from partially observable data is assisted by active learning of ordinary *Markov decision processes* (MDPs) of the underlying environment, learned through the approach of [116], to track the environmental states. The accuracy of the learned MDP is guaranteed through a combination of automata learning, testing, and statistics in [160].

Challenge 18
How can we handle partially observable models? Can we employ specific learning methods or variants of the models encoding possibly missed observations?

Such a question is especially interesting because of its links to non-deterministic models. Indeed, non-determinism can result from partial observability. Non-deterministic models give rise to numerous problems: the number of possible states after a run in a non-deterministic model may increase unbounded during model checking and testing such models gives rise to specific problems [14, 142, 174].

Learning non-deterministic systems is open [170], despite several proposals [31, 56, 102, 139, 181]. The problem of learning non-deterministic systems can also be tackled without use of the MAT framework, integrating ideas from adaptive testing [141].

Challenge 19
Is it possible to learn languages for which no deterministic model exists?

Resets In many existing learning algorithms, it is assumed that the SUL can be *reset*. That is, after each query it can be brought back to its initial state, the starting point of the next query. In practice, however, resetting a system may be prohibitively costly or even impossible. To address this issue, several algorithms have been developed that do not require the SUL to be reset [81, 82, 147]. These approaches either use a *homing sequence* to bring the system back to a known state by providing the right inputs, or determine the current state by means of a *characterising set* of input sequences. A recent extension by Foster *et al.* [71] brings this approach to extended finite state machines. All these algorithms still operate under a number of assumptions, and the practical application is thus not straightforward.

Challenge 20
Can resetless approaches be applied effectively in practice?

6 Theoretical challenges

In this section, we discuss a variety of open problems within the theory and algorithmics of automata learning, focusing on model expressivity, scalability, extensions of the MAT framework and finally on counterexample search and assessment of the quality of learned models.

6.1 Model expressivity

Since the introduction of active automata learning for DFAs, there have been numerous extensions of L^*-type algorithms to other types of (automata) models. A classical extension is to Mealy machines, which provide a natural yet simple model for input-output behaviour, and is standard in automata learning. However, there have been extensions to much richer features such as automata with data and time, to succinct representations, quantitative features, and much more. We do not give a comprehensive overview of these extensions, instead focusing on several research challenges.

Challenge 21
How far does active automata learning extend to efficient inference of rich, multi-faceted systems?

Data languages and register automata A first class of extensions is to *data languages*, which feature infinite alphabets, and are relevant to capture various types of resources and data. A classical automata model recognising these languages is that of *register automata*.

There are several algorithms for active learning of register automata [3, 39, 95] and the related model of nominal automata [126, 127]. This is an ongoing line of research, and a recent result improves on the complexity of learning register automata by using classification tree-based learning algorithms [52]. However, the computational complexity of the underlying tree queries remains an open challenge. At a more technical level, in the existing works, tree queries produce symbolic representations of residual data languages (i.e., fragments of register automata) by executing membership queries for all decisions that could be modelled in a tree, requiring an excessive number of tests. This leads to the following main challenge.

Challenge 22
Can we come up with learning methods that are as lazy as possible to avoid high testing costs that are (sometimes) unnecessary?

Orthogonally, the current approaches are limited to comparing registers for *equality* or for their *order*—the development of learning algorithms for models with different and more expressive tests remains an open question.

Challenge 23
Can we extend learning of register automata beyond (in)equality testing?

The known approaches to learning register automata are able to infer guards w.r.t. data stored in registers, but not comparison to fixed constants. Some methods can be extended to handle a finite set of constants given as parameters to the learning, but none handle the learning of unspecified constants, even for (in)equality theories.

Challenge 24
Can we extend learning of register automata to (unspecified) constants?

Timed systems Active learning of timed systems (i.e., systems whose behaviour depends on the precise timing of actions) has been mostly studied using the timed automaton formalism [13] and subclasses thereof. Timed automata use *clocks* to store the time elapsed since a certain action has occurred, and these clocks can then be compared to each other in guards to allow or disallow a transition.

A strand of research has focused on deterministic one clock timed automata [15], with optimizations using constraint solving [187] or genetic mutations [159]. A subclass of non-deterministic one-clock timed automata, named non-deterministic real-time automata, where the clock is reset on each transition, has also been proven learnable [16].

For more complex models, *event-recording automata* (ERAs) i.e., timed automata with one clock for each action of the alphabet, reset each time this action is met, have been extensively studied [78, 79]. This approach has also been generalised to a superclass of ERAs where a transition may either reset the clock of its action or no clock at all, introducing *invalidity*, a constructive way to detect (im)possible reset choices [83].

An algorithm extending L^* to the complete class of deterministic timed automata has recently been introduced [182], based on a different semantics of timed automata [115]. This algorithm relies on a precise abstraction distinguishing all equivalent behaviours, limiting its practical impact. Alternatively, using the standard semantics, timed automata with multiple clocks can be learned with genetic programming [9].

Challenge 25
How to achieve practical learning algorithms for (deterministic) timed automata, using the formal methods community knowledge on abstraction and efficient algorithms for these models?

Probabilistic and weighted systems We discuss challenges related to learning quantitative systems, which we roughly divide into two parts: theoretical challenges and more practical approximate methods related to model-based reinforcement learning.

There are extensions of learning algorithms to certain forms of probabilistic systems and weighted automata; the two are related, but we start discussing the latter. Weighted

automata are parametric in a semiring that captures the type of quantities involved and the associated algebraic structure. For the case that the semiring is a field, an L^*-type learning algorithm goes back to Bergadano and Varricchio [29], and is well established [25]. There have been proposals for the case that the underlying semiring is that of the integers, with some development in the 2020s [29, 36, 177]; the case of the natural numbers is still open.

Challenge 26
Can we identify theoretical limits of exact automata learning for quantitative types of systems such as weighted automata?

Tappler et al. [162] have extended L^* to *Markov decision processes* (MDPs). More specifically, this work is focused on deterministic MDPs. The probabilities in their MDPs are derived as maximum likelihood estimates, i.e., fractions of observation counts, and do not account for any statistical errors through confidence intervals. The area of learning MDPs is closely related to the field of model-based reinforcement learning; see earlier discussion in Sect. 5.2. When learning probabilistic systems, exact learning is no longer feasible or realistic, leading to the challenge of including approximation in active automata learning algorithms for quantitative systems, and providing suitable *probably approximately correct* (PAC) guarantees [172].

Challenge 27
Is it possible to develop approximate automata learning techniques with PAC guarantees for quantitative systems?

Unifying frameworks In recent years there have been various category-theoretic approaches to automata learning [26, 43, 168, 176, 178], which unify learning algorithms for various types of models. These frameworks differ in the scope of models; they share that they are all based on generalising L^* and its variants. Some of these approaches are particularly suitable for learning *succinct* automata, making it possible to recover, e.g., learning algorithms for *non-deterministic finite automata* (NFAs) [31] and nominal NFAs [126] as instances.

For the specific case of data languages and timed systems, one can notice that these yield similar difficulties in the design of learning algorithms. In fact, both TAs and RAs are automata with guards and memory, with a very similar syntax and (somewhat) closely related semantics. Furthermore, both are related to nominal automata in terms of expressivity, that can be seen as a categorical way of approaching both formalisms. The connection between time and data is especially important to the active learning of CPSs (see Sect. 3.1).

Challenge 28
What is the common ground between learning timed automata and register automata?

An open problem in this context is to provide a general categorical account of *efficient* learning algorithms such as TTT [96] and $L^\#$ [171]. Towards this challenge, in particular, in the direction of TTT, the work of Isberner provides a more operational view of some of the components that need to be implemented for such an effort [94].

Challenge 29
Can we establish a unifying framework for efficient automata learning?

Finally, given the many extensions of automata learning, a further problem is to *combine* these extensions, ideally through a modular framework.

Challenge 30
Is a modular approach to extensions of expressivity feasible?

6.2 Scalability

A major challenge common to all methods for automata learning is *scalability*, i.e., the runtime and/or memory usage typically increases quickly with the size of the automaton that is to be learned. On the theoretical side, there are several negative results showing infeasibility of certain approaches [17, 19, 20, 76]. In particular, finite state machines cannot be learned in polynomial time with only membership queries or only equivalence queries. Furthermore, in the setting of passive learning, the problem of finding a smallest automaton that agrees with a given sample of positive/negative words is NP-complete. On top of this, in practice even polynomial learning algorithms may suffer from scalability issues since *explicit* (i.e., state-by-state) representations of a system's state space are often very large, for example, because it individually captures each element of the data domain it operates on.

The problem of scalability and *state-space explosion* [173] is well known in the field of formal verification, and many advanced methods have been developed in response. We explore a few of these techniques and how their application may benefit automata learning.

Challenge 31
How can we improve scalability by employing formal methods tactics (abstraction, decomposition) or by employing machine learning?

Abstraction As discussed before, the introduction of complex environments or data can pose a large challenge in the context of automata learning. *Abstraction* is a widespread approach to deal with this by only considering a certain part or aspect of a system. Several successful approaches exist for learning automata with data, time or probabilities, see the discussions in Sect. 6.1. However, abstraction can also be beneficial in other scenarios, in particular when the level of abstraction required is determined automatically. For example, the tool Tomte [3] derives a mapping from an abstract alphabet to real-world actions with *counter-example guided abstraction refinement* (CEGAR). Similar refinement techniques are applied by Howar et al. [90] to automatically derive an abstract alphabet that yields a deterministic representation of the system. Still, non-determinism, often perceived as noise, remains challenging in practice (see Sect. 5.2). Thus extending these approaches with the ability to include abstraction over multiple aspects of a system may benefit applications of automata learning.

Compositional techniques In compositional verification, the approach is to consider individual components of the system and lift the results of that analysis to the system level. In the case of concurrent components, compositional verification may avoid the exponential blow-up of computing the state space of their interleaving. Common challenges for such techniques include: how to obtain sufficient information when considering only a subset of the components and how to translate any learned facts between component and system level.

A few compositional approaches for learning already exist. *Product automata* are a type of Mealy machine where the output is the combination of outputs of several Mealy machines; these components can be learned individually [125]. Learning which parallel components make up a system can be done automatically [109]. However, these two techniques do not support interacting components. Systems consisting of synchronising automata can be learned compositionally, either with a priori knowledge of the individual alphabets [133] or fully automatically [84]. Finally, active learning can also be applied to *systems of procedural automata* [74], a collection of automata that can call each other in a way similar to procedure calls. All these compositional approaches show significant performance gains over monolithic approaches.

Although compositional techniques show promise, more work is required to make them more black box and applicable to a wider range of systems, including rich formalisms that support various ways of composition. Furthermore, the application of these techniques in a real-world context is not straightforward and requires more study.

Challenge 32
How can systems be learned compositionally in a fully black-box manner?

Machine learning Machine learning techniques can also be applied in conjunction with automata learning to address the scalability problem.

For example, mappers can also be defined or refined by machine learning: the fingerprinting technique presented by Sabahi-Kaviani et al. [148] also explores the use of machine learning to identify sequences of packets that occur together, allowing to greatly reduce the size of the alphabet on which the Learner operates.

Machine learning can also be used to reduce the size of the necessary sample set. For example, learning an MDP model of an environment controlled by an agent generated through deep learning is generally too demanding for automata learning techniques, but has been achieved [164] by combining passive learning, RL techniques and machine learning techniques (dimensionality reduction and clustering), leading to an MDP model of the controlled environment. This resulting model is iteratively fine-tuned using active sampling.

Another approach of this approximation principle is mapping the active learning problem to the problem of finding a specific *recurrent neural network* (RNN) architecture that is able to learn a Mealy machine from the given data [10]. An automaton is extracted from the trained RNN model with the proposed architecture. The RNN model is trained through a specific constrained approach exploiting a regularization that ensures certainty about the encoded automaton in the RNN model. The evaluation results show that the proposed RNN architecture and training approach performs well in comparison with L^* when the dataset is relatively small with low coverage of system behaviour.

6.3 Finding counterexamples and assessing quality

With very efficient learning algorithms, the approximation of equivalence queries becomes the bottleneck in black-box scenarios. The task of most equivalence queries during learning is to produce counterexamples. Towards the end of the learning process, the focus may change towards testing the quality (e.g., accuracy) of the produced hypothesis models.

For approximating equivalence queries, currently most works use randomised generic model-based testing or conformance testing methods [35, 169]. There is a range of different conformance testing techniques that can be used—we refer to Aichernig et al. [11] and Garhewal and Damasceno [75] for recent works that experimentally compare many of these techniques for their performance in the context of learning, and to Dorofeeva et al. [53] for an earlier comparison that specifically focuses on conformance testing itself.

The underlying methods aim at testing conformance, not at finding counterexamples. While both these perspectives are related, the strategies for selecting individual tests may be very different. Generic testing methods, e.g., are not designed to use information that is reflected in the hypothesis already. Moreover, there are various situations where the learner has access to additional information beyond the regular MAT and conformance testing setting. For instance, while the MAT framework assumes the SUL to be a black-box, in practice there can be various levels of access to the internals of the SUL. One idea is to use (limited) access to code to guide the search for counterexamples. Another source of information is execution logs or traces, which can be integrated in active learning [188]. Other recent approaches aim at reducing the size of conformance test suites by discovering and exploiting structure in the hypothesis and in real-world systems [106] or by applying ideas from mutation testing [5].

Challenge 33
Are there general approaches for finding counterexamples fast in practice?

While active learning and testing are already very tightly integrated, as the equivalence query is typically implemented by testing, at a theoretical level their interaction can be further refined. This correspondence is explored in, e.g., by Aichernig et al. [6] and Berg et al. [28]. A framework replacing MAT to ease the integration of testing (and other) methods is proposed by Ferreira et al. [62]. The $L^{\#}$ algorithm uses an observation tree collecting all tests as its primary data structure, following the earlier use of observation trees in learning [156]. We believe that there is still much insight to gain from the study of the fundamental connections between learning and testing.

Challenge 34
How can we further integrate automata learning with testing techniques?

Assessing the quality of learned hypotheses Model-based approaches are directly influenced by the quality of the model. Assessing the quality of models *during* active automata learning is still largely an open problem [170].

Challenge 35
Can we provide measures of accuracy of the learned model with respect to the SUL?

Two works that made initial contributions in this direction introduce metrics of progress [175] and (in a passive learning scenario) use optimization to balance over-approximation and under-approximation [166].

7 Conclusion & outlook

We have identified and discussed a variety of challenges in the field of active automata learning. We put forward some general recommendations that we think will help attack these challenges and propel the field as a whole. First, the field currently lacks a uniform benchmarking infrastructure and a standardised set of benchmark models (apart from the models on the *Automata Wiki* [134]). This makes it necessary to redo experiments to obtain baselines for new contributions and difficult to compare results.

Furthermore, the fields of SAT/SMT-solving, software verification and model checking have each benefitted from successful annual competitions. We think the automata learning community would likewise benefit from the revival of past competitions such as ZULU [44] and PAutomaC [179]. A single well-recognised event would create a focal point for the community and enable frequent exchange and the coordination of joint efforts.

Funding information T. Neele is supported by NWO grant VI.Veni.232.224. M. Suilen is supported by NWO grant OCENW.KLEIN.187. J. Rot is supported by NWO grant VI.Vidi.223.096.

References

1. Aarts, F., Schmaltz, J., Vaandrager, F.W.: Inference and abstraction of the biometric passport. In: ISoLA (1). Lecture Notes in Computer Science, vol. 6415, pp. 673–686. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-16558-0_54
2. Aarts, F., de Ruiter, J., Poll, E.: Formal models of bank cards for free. In: ICST Workshops, pp. 461–468. IEEE Computer Society (2013). https://doi.org/10.1109/ICSTW.2013.60
3. Aarts, F., Fiterau-Brostean, P., Kuppens, H., Vaandrager, F.W.: Learning register automata with fresh value generation. In: ICTAC. Lecture Notes in Computer Science, vol. 9399, pp. 165–183. Springer, Berlin (2015). https://doi.org/10.1007/978-3-319-25150-9_11
4. Abel, A., Reineke, J.: Gray-box learning of serial compositions of Mealy machines. In: NFM. Lecture Notes in Computer Science, vol. 9690, pp. 272–287. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-40648-0_21
5. Aichernig, B.K., Tappler, M.: Efficient active automata learning via mutation testing. J. Autom. Reason. **63**(4), 1103–1134 (2019). https://doi.org/10.1007/S10817-018-9486-0

6. Aichernig, B.K., Mostowski, W., Mousavi, M.R., Tappler, M., Taromirad, M.: Model learning and model-based testing. In: Machine Learning for Dynamic Software Analysis. Lecture Notes in Computer Science, vol. 11026, pp. 74–100. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-96562-8_3
7. Aichernig, B.K., Bloem, R., Ebrahimi, M., Horn, M., Pernkopf, F., Roth, W., Rupp, A., Tappler, M., Tranninger, M.: Learning a behavior model of hybrid systems through combining model-based testing and machine learning. In: ICTSS. Lecture Notes in Computer Science, vol. 11812, pp. 3–21. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-31280-0_1
8. Aichernig, B.K., Burghard, C., Korosec, R.: Learning-based testing of an industrial measurement device. In: NFM. Lecture Notes in Computer Science, vol. 11460, pp. 1–18. Springer, Berlin (2019)
9. Aichernig, B.K., Pferscher, A., Tappler, M.: From passive to active: learning timed automata efficiently. In: NFM. Lecture Notes in Computer Science, vol. 12229, pp. 1–19. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-55754-6_1
10. Aichernig, B.K., König, S., Mateis, C., Pferscher, A., Schmidt, D., Tappler, M.: Constrained training of recurrent neural networks for automata learning. In: SEFM. Lecture Notes in Computer Science, vol. 13550, pp. 155–172. Springer, Berlin (2022). https://doi.org/10.1007/978-3-031-17108-6_10
11. Aichernig, B.K., Tappler, M., Wallner, F.: Benchmarking combinations of learning and testing algorithms for automata learning. Form. Asp. Comput. **36**(1), 3:1–3:37 (2024). https://doi.org/10.1145/3605360
12. Ali, S., Sun, H., Zhao, Y.: Model learning: a survey of foundations, tools and applications. Front. Comput. Sci. **15**(5), 155210 (2021). https://doi.org/10.1007/S11704-019-9212-Z
13. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
14. Alur, R., Courcoubetis, C., Yannakakis, M.: Distinguishing tests for nondeterministic and probabilistic machines. In: STOC, pp. 363–372. ACM, New York (1995). https://doi.org/10.1145/225058.225161
15. An, J., Chen, M., Zhan, B., Zhan, N., Zhang, M.: Learning one-clock timed automata. In: TACAS (1). Lecture Notes in Computer Science, vol. 12078, pp. 444–462. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-45190-5_25
16. An, J., Wang, L., Zhan, B., Zhan, N., Zhang, M.: Learning real-time automata. Sci. China Inf. Sci. **64**(9) (2021). https://doi.org/10.1007/S11432-019-2767-4
17. Angluin, D.: A note on the number of queries needed to identify regular languages. Inf. Control **51**(1), 76–87 (1981). https://doi.org/10.1016/S0019-9958(81)90090-5
18. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6
19. Angluin, D.: Negative results for equivalence queries. Mach. Learn. **5**, 121–150 (1990). https://doi.org/10.1007/BF00116034
20. Angluin, D.: Queries revisited. Theor. Comput. Sci. **313**(2), 175–194 (2004). https://doi.org/10.1016/J.TCS.2003.11.004
21. Angluin, D., Laird, P.D.: Learning from noisy examples. Mach. Learn. **2**(4), 343–370 (1987). https://doi.org/10.1007/BF00116829
22. Arif, M.F., Larraz, D., Echeverria, M., Reynolds, A., Chowdhury, O., Tinelli, C.: SYSLITE: syntax-guided synthesis of PLTL formulas from finite traces. In: FMCAD, pp. 93–103. IEEE (2020). https://doi.org/10.34727/2020/ISBN.978-3-85448-042-6_16
23. Aslam, K., Cleophas, L., Schiffelers, R.R.H., van den Brand, M.: Interface protocol inference to aid understanding legacy software components. Softw. Syst. Model. **19**(6), 1519–1540 (2020). https://doi.org/10.1007/S10270-020-00809-2
24. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
25. Balle, B., Mohri, M.: Learning weighted automata. In: CAI. Lecture Notes in Computer Science, vol. 9270, pp. 1–21. Springer, Berlin (2015). https://doi.org/10.1007/978-3-319-23021-4_1
26. Barlocco, S., Kupke, C., Rot, J.: Coalgebra learning via duality. In: FoSSaCS. Lecture Notes in Computer Science, vol. 11425, pp. 62–79. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-17127-8_4
27. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification - Introductory and Advanced Topics. Lecture Notes in Computer Science, vol. 10457. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-75632-5
28. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the correspondence between conformance testing and regular inference. In: FASE. Lecture Notes in Computer Science, vol. 3442, pp. 175–189. Springer, Berlin (2005). https://doi.org/10.1007/978-3-540-31984-9_14
29. Bergadano, F., Varricchio, S.: Learning behaviors of automata from multiplicity and equivalence queries. SIAM J. Comput. **25**(6), 1268–1280 (1996). https://doi.org/10.1137/S009753979326091X
30. Blumreiter, M., Greenyer, J., Garcia, F.J.C., Klös, V., Schwammberger, M., Sommer, C., Vogelsang, A., Wortmann, A.: Towards self-explainable cyber-physical systems. In: MoDELS (Companion), pp. 543–548. IEEE (2019). https://doi.org/10.1109/MODELS-C.2019.00084
31. Bollig, B., Habermehl, P., Kern, C., Leucker, M.: Angluin-style learning of NFA. In: IJCAI, pp. 1004–1009 (2009)
32. Bollig, B., Katoen, J., Kern, C., Leucker, M., Neider, D., Piegdon, D.R.: libalf: the automata learning framework. In: CAV. Lecture Notes in Computer Science, vol. 6174, pp. 360–364. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-14295-6_32
33. Brafman, R.I., Giacomo, G.D.: Regular decision processes. Artif. Intell. **331**, 104113 (2024). https://doi.org/10.1016/J.ARTINT.2024.104113
34. Bras, P.L., Robb, D.A., Methven, T.S., Padilla, S., Chantler, M.J.: Improving user confidence in concept maps: exploring data driven explanations. In: CHI, p. 404. ACM, New York (2018). https://doi.org/10.1145/3173574.3173978
35. Broy, M., Jonsson, B., Katoen, J., Leucker, M., Pretschner, A. (eds.): Model-Based Testing of Reactive Systems, Advanced Lectures [The Volume Is the Outcome of a Research Seminar That Was Held in Schloss Dagstuhl in January 2004]. Lecture Notes in Computer Science, vol. 3472. Springer, Berlin (2005). https://doi.org/10.1007/B137241
36. Buna-Marginean, A., Cheval, V., Shirmohammadi, M., Worrell, J.: On learning polynomial recursive programs. Proc. ACM Program. Lang. **8**(POPL), 1001–1027 (2024). https://doi.org/10.1145/3632876
37. Cândido, J., Aniche, M., van Deursen, A.: Log-based software monitoring: a systematic mapping study. PeerJ Comput. Sci. **7**, e489 (2021). https://doi.org/10.7717/PEERJ-CS.489
38. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: ICGI. Lecture Notes in Computer Science, vol. 862, pp. 139–152. Springer, Berlin (1994)
39. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Active learning for extended finite state machines. Form. Asp. Comput. **28**(2), 233–263 (2016). https://doi.org/10.1007/S00165-016-0355-5
40. Chaki, S., Clarke, E.M., Sharygina, N., Sinha, N.: Verification of evolving software via component substitutability analysis. Form. Methods Syst. Des. **32**(3), 235–266 (2008). https://doi.org/10.1007/S10703-008-0053-X
41. Chalupar, G., Peherstorfer, S., Poll, E., de Ruiter, J.: Automated reverse engineering using Lego®. In: WOOT. USENIX Association (2014)

42. Clarke, E.M., Emerson, E.A., Sifakis, J.: Model checking: algorithmic verification and debugging. Commun. ACM **52**(11), 74–84 (2009). https://doi.org/10.1145/1592761.1592781
43. Colcombet, T., Petrisan, D., Stabile, R.: Learning automata and transducers: a categorical approach. In: CSL. LIPIcs, vol. 183, pp. 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPICS.CSL.2021.15
44. Combe, D., de la Higuera, C., Janodet, J.: Zulu: an interactive learning competition. In: FSMNLP. Lecture Notes in Computer Science, vol. 6062, pp. 139–146. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-14684-8_15
45. Cordy, M., Devroey, X., Legay, A., Perrouin, G., Classen, A., Heymans, P., Schobbens, P., Raskin, J.: A decade of featured transition systems. In: From Software Engineering to Formal Methods and Tools, and Back. Lecture Notes in Computer Science, vol. 11865, pp. 285–312. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-30985-5_18
46. Damasceno, C.D.N., Mousavi, M.R., da Silva Simão, A.: Learning to reuse: adaptive model learning for evolving systems. In: IFM. Lecture Notes in Computer Science, vol. 11918, pp. 138–156. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-34968-4_8
47. Damasceno, C.D.N., Mousavi, M.R., da Silva Simão, A.: Learning by sampling: learning behavioral family models from software product lines. Empir. Softw. Eng. **26**(1), 4 (2021). https://doi.org/10.1007/S10664-020-09912-W
48. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, Cambridge (2010). https://doi.org/10.1017/CBO9781139194655
49. de Ruiter, J., Poll, E.: Protocol state fuzzing of TLS implementations. In: USENIX Security Symposium, pp. 193–206. USENIX Association (2015)
50. Derler, P., Lee, E.A., Sangiovanni-Vincentelli, A.L.: Modeling cyber-physical systems. Proc. IEEE **100**(1), 13–28 (2012). https://doi.org/10.1109/JPROC.2011.2160929
51. Dierl, S., Howar, F.M., Kauffman, S., Kristjansen, M., Larsen, K.G., Lorber, F., Mauritz, M.: Learning symbolic timed models from concrete timed data. In: NFM. Lecture Notes in Computer Science, vol. 13903, pp. 104–121. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-33170-1_7
52. Dierl, S., Fiterau-Brostean, P., Howar, F., Jonsson, B., Sagonas, K., Tåquist, F.: Scalable tree-based register automata learning. In: TACAS (2). Lecture Notes in Computer Science, vol. 14571, pp. 87–108. Springer, Berlin (2024). https://doi.org/10.1007/978-3-031-57249-4_5
53. Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A.R., Yevtushenko, N.: FSM-based conformance testing methods: a survey annotated with experimental evaluation. Inf. Softw. Technol. **52**(12), 1286–1297 (2010). https://doi.org/10.1016/J.INFSOF.2010.07.001
54. Dupont, P., Lambeau, B., Damas, C., van Lamsweerde, A.: The QSM algorithm and its application to software behavior model induction. Appl. Artif. Intell. **22**(1&2), 77–115 (2008)
55. Eddy, W.: RFC 9293: Transmission Control Protocol (TCP). Internet Engineering Task Force (2022)
56. El-Fakih, K., Groz, R., Irfan, M.N., Shahbaz, M.: Learning finite state models of observable nondeterministic systems in a testing context. In: 22nd IFIP International Conference on Testing Software and Systems, pp. 97–102 (2010)
57. Farooqui, A., Fabian, M.: Synthesis of supervisors for unknown plant models using active learning. In: CASE, pp. 502–508. IEEE (2019). https://doi.org/10.1109/COASE.2019.8843177
58. Farooqui, A., Claase, R.T., Fabian, M.: On active learning for supervisor synthesis. IEEE Trans. Autom. Sci. Eng. **21**(1), 78–90 (2024). https://doi.org/10.1109/TASE.2022.3216759
59. Feng, L., Kwiatkowska, M.Z., Parker, D.: Compositional verification of probabilistic systems using learning. In: QEST, pp. 133–142. IEEE Computer Society (2010). https://doi.org/10.1109/QEST.2010.24
60. Ferreira, T., Brewton, H., D'Antoni, L., Silva, A.: Prognosis: closed-box analysis of network protocol implementations. In: SIGCOMM, pp. 762–774. ACM, New York (2021). https://doi.org/10.1145/3452296.3472938
61. Ferreira, T., van Heerdt, G., Silva, A.: Tree-based adaptive model learning. In: A Journey from Process Algebra via Timed Automata to Model Learning. Lecture Notes in Computer Science, vol. 13560, pp. 164–179. Springer, Berlin (2022). https://doi.org/10.1007/978-3-031-15629-8_10
62. Ferreira, T., Henry, L., da Silva, R.F., Silva, A.: Conflict-aware active automata learning. In: GandALF. EPTCS, vol. 390, pp. 150–167 (2023). https://doi.org/10.4204/EPTCS.390.10
63. Fiterau-Brostean, P., Howar, F.: Learning-based testing the sliding window behavior of TCP implementations. In: FMICS-AVoCS. Lecture Notes in Computer Science, vol. 10471, pp. 185–200. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-67113-0_12
64. Fiterau-Brostean, P., Janssen, R., Vaandrager, F.W.: Combining model learning and model checking to analyze TCP implementations. In: CAV (2). Lecture Notes in Computer Science, vol. 9780, pp. 454–471. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-41540-6_25
65. Fiterau-Brostean, P., Lenaerts, T., Poll, E., de Ruiter, J., Vaandrager, F.W., Verleg, P.: Model learning and model checking of SSH implementations. In: SPIN, pp. 142–151. ACM, New York (2017). https://doi.org/10.1145/3092282.3092289
66. Fortz, S.: Learning featured transition systems. Ph.D. thesis, University of Namur (2023). https://researchportal.unamur.be/en/studentTheses/learning-featured-transition-systems
67. Fortz, S., Temple, P., Devroey, X., Heymans, P., Perrouin, G.: VaryMinions: leveraging RNNs to identify variants in event logs. In: MaLTeSQuE@ESEC/SIGSOFT FSE, pp. 13–18. ACM, New York (2021). https://doi.org/10.1145/3472674.3473980
68. Fortz, S., Temple, P., Devroey, X., Heymans, P., Perrouin, G.: VaryMinions: leveraging RNNs to identify variants in variability-intensive systems' logs. Empir. Softw. Eng. **29**(4), 99 (2024). https://doi.org/10.1007/S10664-024-10473-5
69. Fortz, S., Temple, P., Devroey, X., Perrouin, G.: Towards feature-based ML-enabled behaviour location. In: VaMoS, pp. 152–154. ACM, New York (2024). https://doi.org/10.1145/3634713.3634734
70. Foster, M., Derrick, J., Walkinshaw, N.: Reverse-engineering EFSMs with data dependencies. In: ICTSS. Lecture Notes in Computer Science, vol. 13045, pp. 37–54. Springer, Berlin (2021)
71. Foster, M., Groz, R., Oriat, C., da Silva Simão, A., Vega, G., Walkinshaw, N.: Active inference of EFSMs without reset. In: ICFEM. Lecture Notes in Computer Science, vol. 14308, pp. 29–46. Springer, Berlin (2023). https://doi.org/10.1007/978-981-99-7584-6_3
72. Fragal, V.H., Simão, A., Mousavi, M.R.: Validated test models for software product lines: featured finite state machines. In: FACS. Lecture Notes in Computer Science, vol. 10231, pp. 210–227 (2016). https://doi.org/10.1007/978-3-319-57666-4_13
73. Frenkel, H., Grumberg, O., Pasareanu, C.S., Sheinvald, S.: Assume, guarantee or repair. In: TACAS (1). Lecture Notes in Computer Science, vol. 12078, pp. 211–227. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-45190-5_12
74. Frohme, M., Steffen, B.: Compositional learning of mutually recursive procedural systems. Int. J. Softw. Tools Technol. Transf. **23**(4), 521–543 (2021). https://doi.org/10.1007/S10009-021-00634-Y

75. Garhewal, B., Damasceno, C.D.N.: An experimental evaluation of conformance testing techniques in active automata learning. In: MODELS, pp. 217–227. IEEE (2023). https://doi.org/10.1109/MODELS58315.2023.00012
76. Gold, E.M.: Complexity of automaton identification from given data. Inf. Control **37**(3), 302–320 (1978). https://doi.org/10.1016/S0019-9958(78)90562-4
77. Greenyer, J., Lochau, M., Vogel, T.: Explainable software for cyber-physical systems (ES4CPS): report from the GI Dagstuhl seminar 19023, January 06-11 2019, Schloss Dagstuhl (2019). CoRR arXiv:1904.11851
78. Grinchtein, O.: Learning of timed systems. Ph.D. thesis, Uppsala University, Sweden (2008)
79. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. Theor. Comput. Sci. **411**(47), 4029–4054 (2010). https://doi.org/10.1016/J.TCS.2010.07.008
80. Groce, A., Peled, D.A., Yannakakis, M.: Adaptive model checking. Log. J. IGPL **14**(5), 729–744 (2006). https://doi.org/10.1093/JIGPAL/JZL007
81. Groz, R., da Silva Simão, A., Petrenko, A., Oriat, C.: Inferring finite state machines without reset using state identification sequences. In: ICTSS. Lecture Notes in Computer Science, vol. 9447, pp. 161–177. Springer, Berlin (2015). https://doi.org/10.1007/978-3-319-25945-1_10
82. Groz, R., Brémond, N., da Silva Simão, A., Oriat, C.: hW-inference: a heuristic approach to retrieve models through black box testing. J. Syst. Softw. **159** (2020). https://doi.org/10.1016/J.JSS.2019.110426
83. Henry, L., Jéron, T., Markey, N.: Active learning of timed automata with unobservable resets. In: FORMATS. Lecture Notes in Computer Science, vol. 12288, pp. 144–160. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-57628-8_9
84. Henry, L., Mousavi, M.R., Neele, T., Sammartino, M.: Compositional active learning of synchronizing systems through automated alphabet refinement. In: CONCUR. LIPIcs, vol. 348, pp. 20:1–20:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2025). https://doi.org/10.4230/LIPICS.CONCUR.2025.20
85. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker Storm. Int. J. Softw. Tools Technol. Transf. **24**(4), 589–610 (2022). https://doi.org/10.1007/S10009-021-00633-Z
86. Henzinger, T.A., Sifakis, J.: The embedded systems design challenge. In: FM. Lecture Notes in Computer Science, vol. 4085, pp. 1–15. Springer, Berlin (2006). https://doi.org/10.1007/11813040_1
87. Holzinger, A., Saranti, A., Molnar, C., Biecek, P., Samek, W.: Explainable AI methods - a brief overview. In: xxAI@ICML. Lecture Notes in Computer Science, vol. 13200, pp. 13–38. Springer, Berlin (2020). https://doi.org/10.1007/978-3-031-04083-2_2
88. Howar, F., Steffen, B.: Active automata learning in practice - an annotated bibliography of the years 2011 to 2016. In: Machine Learning for Dynamic Software Analysis. Lecture Notes in Computer Science, vol. 11026, pp. 123–148. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-96562-8_5
89. Howar, F., Steffen, B.: Active automata learning as black-box search and lazy partition refinement. In: A Journey from Process Algebra via Timed Automata to Model Learning. Lecture Notes in Computer Science, vol. 13560, pp. 321–338. Springer, Berlin (2022). https://doi.org/10.1007/978-3-031-15629-8_17
90. Howar, F., Steffen, B., Merten, M.: Automata learning with automated alphabet abstraction refinement. In: VMCAI. Lecture Notes in Computer Science, vol. 6538, pp. 263–277. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-18275-4_19
91. Howar, F., Merten, M., Steffen, B., Margaria, T.: Practical Aspects of Active Automata Learning, Chap. 11, pp. 235–267. John Wiley & Sons, Ltd (2012). https://doi.org/10.1002/9781118459898.ch11
92. Huistra, D., Meijer, J., van de Pol, J.: Adaptive learning for learn-based regression testing. In: FMICS. Lecture Notes in Computer Science, vol. 11119, pp. 162–177. Springer, Berlin (2018). https://doi.org/10.1007/978-3-030-00244-2_11
93. Irfan, M., Oriat, C., Groz, R.: Model inference and testing. Adv. Comput. **89**, 89–139 (2013). https://doi.org/10.1016/B978-0-12-408094-2.00003-5
94. Isberner, M.: Foundations of active automata learning: an algorithmic perspective. Ph.D. thesis, Technical University Dortmund, Germany (2015). http://hdl.handle.net/2003/34282
95. Isberner, M., Howar, F., Steffen, B.: Learning register automata: from languages to program structures. Mach. Learn. **96**(1–2), 65–98 (2014). https://doi.org/10.1007/S10994-013-5419-7
96. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: a redundancy-free approach to active automata learning. In: RV. Lecture Notes in Computer Science, vol. 8734, pp. 307–322. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-11164-3_26
97. Isberner, M., Howar, F., Steffen, B.: The open-source LearnLib - a framework for active automata learning. In: CAV (1). Lecture Notes in Computer Science, vol. 9206, pp. 487–495. Springer, Berlin (2015). https://doi.org/10.1007/978-3-319-21690-4_32
98. Jard, C., Jéron, T.: TGV: theory, principles and algorithms. Int. J. Softw. Tools Technol. Transf. **7**(4), 297–315 (2005). https://doi.org/10.1007/S10009-004-0153-X
99. Jasper, M., Mues, M., Murtovi, A., Schlüter, M., Howar, F., Steffen, B., Schordan, M., Hendriks, D., Schiffelers, R.R.H., Kuppens, H., Vaandrager, F.W.: RERS 2019: combining synthesis with real-world models. In: TACAS (3). Lecture Notes in Computer Science, vol. 11429, pp. 101–115. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-17502-3_7
100. Jaulmes, R., Pineau, J., Precup, D.: Active learning in partially observable Markov decision processes. In: ECML. Lecture Notes in Computer Science, vol. 3720, pp. 601–608. Springer, Berlin (2005). https://doi.org/10.1007/11564096_59
101. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
102. Khalili, A., Tacchella, A.: Learning nondeterministic Mealy machines. In: ICGI. JMLR Workshop and Conference Proceedings, vol. 34, pp. 109–123. JMLR.org (2014)
103. Khmelnitsky, I., Haddad, S., Ye, L., Barbot, B., Bollig, B., Leucker, M., Neider, D., Roy, R.: Analyzing robustness of angluin's L* algorithm in presence of noise. In: GandALF. EPTCS, vol. 370, pp. 81–96 (2022). https://doi.org/10.4204/EPTCS.370.6
104. Khosrowjerdi, H., Meinke, K., Rasmusson, A.: Learning-based testing for safety critical automotive applications. In: IMBSA. Lecture Notes in Computer Science, vol. 10437, pp. 197–211. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-64119-5_13
105. Kleene, S.C.: Representation of events in nerve nets and finite automata. Tech. Rep., RAND Corporation, Santa Monica, CA (1951)
106. Kruger, L., Junges, S., Rot, J.: Small test suites for active automata learning. In: TACAS (2). Lecture Notes in Computer Science, vol. 14571, pp. 109–129. Springer, Berlin (2024). https://doi.org/10.1007/978-3-031-57249-4_6
107. Kruger, L., Junges, S., Rot, J.: State matching and multiple references in adaptive active automata learning. In: FM (1). Lecture Notes in Computer Science, vol. 14933, pp. 267–284. Springer, Berlin (2024). https://doi.org/10.1007/978-3-031-71162-6_14
108. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-22110-1_47

109. Labbaf, F., Groote, J.F., Hojjat, H., Mousavi, M.R.: Compositional learning for interleaving parallel automata. In: FoSSaCS. Lecture Notes in Computer Science, vol. 13992, pp. 413–435. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-30829-1_20
110. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Methods Program. **78**(5), 293–303 (2009). https://doi.org/10.1016/J.JLAP.2008.08.004
111. Lim, B.Y., Dey, A.K., Avrahami, D.: Why and why not explanations improve the intelligibility of context-aware intelligent systems. In: CHI, pp. 2119–2128. ACM, New York (2009). https://doi.org/10.1145/1518701.1519023
112. Lin, S., André, É., Liu, Y., Sun, J., Dong, J.S.: Learning assumptions for compositional verification of timed systems. IEEE Trans. Softw. Eng. **40**(2), 137–153 (2014). https://doi.org/10.1109/TSE.2013.57
113. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In: AAAI/IAAI, pp. 541–548. AAAI Press / The MIT Press (1999)
114. Maler, O., Pnueli, A.: On the learnability of infinitary regular sets. Inf. Comput. **118**(2), 316–326 (1995). https://doi.org/10.1006/INCO.1995.1070
115. Maler, O., Pnueli, A.: On recognizable timed languages. In: FoSSaCS. Lecture Notes in Computer Science, vol. 2987, pp. 348–362. Springer, Berlin (2004). https://doi.org/10.1007/978-3-540-24727-2_25
116. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning deterministic probabilistic automata from a model checking perspective. Mach. Learn. **105**(2), 255–299 (2016). https://doi.org/10.1007/S10994-016-5565-9
117. Marzani, F., Ghassemi, F., Sabahi-Kaviani, Z., van Ede, T., van Steen, M.: Mobile app fingerprinting through automata learning and machine learning. In: IFIP Networking, pp. 1–9. IEEE (2023). https://doi.org/10.23919/IFIPNETWORKING57963.2023.10186420
118. Mayr, F., Yovine, S.: Regular inference on artificial neural networks. In: CD-MAKE. Lecture Notes in Computer Science, vol. 11015, pp. 350–369. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-99740-7_25
119. McCallum, R.A.: Overcoming incomplete perception with utile distinction memory. In: ICML, pp. 190–196. Morgan Kaufmann, San Mateo (1993). https://doi.org/10.1016/B978-1-55860-307-3.50031-9
120. McCallum, R.A.: Instance-based state identification for reinforcement learning. In: NIPS, pp. 377–384. MIT Press, Cambridge (1994)
121. McCallum, R.A.: Instance-based utile distinctions for reinforcement learning with hidden state. In: ICML, pp. 387–395. Morgan Kaufmann, San Mateo (1995). https://doi.org/10.1016/B978-1-55860-377-6.50055-4
122. Mens, T., Gorp, P.V.: A taxonomy of model transformation. In: GRaMoT@GPCE. Electronic Notes in Theoretical Computer Science, vol. 152, pp. 125–142. Elsevier, Amsterdam (2005). https://doi.org/10.1016/J.ENTCS.2005.10.021
123. Merten, M., Steffen, B., Howar, F., Margaria, T.: Next generation LearnLib. In: TACAS. Lecture Notes in Computer Science, vol. 6605, pp. 220–223. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-19835-9_18
124. Moerland, T.M., Broekens, J., Plaat, A., Jonker, C.M.: Model-based reinforcement learning: a survey. Found. Trends Mach. Learn. **16**(1), 1–118 (2023). https://doi.org/10.1561/2200000086
125. Moerman, J.: Learning product automata. In: ICGI. Proceedings of Machine Learning Research, vol. 93, pp. 54–66. PMLR (2018)
126. Moerman, J., Sammartino, M.: Residuality and learning for nondeterministic nominal automata. Log. Methods Comput. Sci. **18**(1) (2022). https://doi.org/10.46298/LMCS-18(1:29)2022
127. Moerman, J., Sammartino, M., Silva, A., Klin, B., Szynwelski, M.: Learning nominal automata. In: POPL, pp. 613–625. ACM, New York (2017). https://doi.org/10.1145/3009837.3009879
128. Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., Peters, J.: Robust reinforcement learning: a review of foundations and recent advances. Mach. Learn. Knowl. Extr. **4**(1), 276–315 (2022). https://doi.org/10.3390/MAKE4010013
129. Mousavi, M.R.: Automata learning for dynamic software product lines: a tutorial. In: SPLC (A), pp. 271–272. ACM, New York (2023). https://doi.org/10.1145/3579027.3609001
130. Muskardin, E., Aichernig, B.K., Pill, I., Pferscher, A., Tappler, M.: AALpy: an active automata learning library. Innov. Syst. Softw. Eng. **18**(3), 417–426 (2022). https://doi.org/10.1007/S11334-022-00449-3
131. Muskardin, E., Aichernig, B.K., Pill, I., Tappler, M.: Learning finite state models from recurrent neural networks. In: IFM. Lecture Notes in Computer Science, vol. 13274, pp. 229–248. Springer, Berlin (2022). https://doi.org/10.1007/978-3-031-07727-2_13
132. Muskardin, E., Tappler, M., Aichernig, B.K., Pill, I.: Reinforcement learning under partial observability guided by learned environment models. In: iFM. Lecture Notes in Computer Science, vol. 14300, pp. 257–276. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-47705-8_14
133. Neele, T., Sammartino, M.: Compositional automata learning of synchronous systems. In: FASE. Lecture Notes in Computer Science, vol. 13991, pp. 47–66. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-30826-0_3
134. Neider, D., Smetsers, R., Vaandrager, F.W., Kuppens, H.: Benchmarks for automata learning and conformance testing. In: Models, Mindsets, Meta. Lecture Notes in Computer Science, vol. 11200, pp. 390–416. Springer, Berlin (2018). https://doi.org/10.1007/978-3-030-22348-9_23
135. Nerode, A.: Linear automaton transformations. Proc. Am. Math. Soc. **9**(4), 541–544 (1958)
136. Niggemann, O., Lohweg, V.: On the diagnosis of cyber-physical production systems. In: AAAI, pp. 4119–4126. AAAI Press, Menlo Park (2015). https://doi.org/10.1609/AAAI.V29I1.9762
137. Omlin, C.W., Giles, C.L.: Extraction of rules from discrete-time recurrent neural networks. Neural Netw. **9**(1), 41–52 (1996). https://doi.org/10.1016/0893-6080(95)00086-0
138. Oncina, J., García, P.: Inferring regular languages in polynomial updated time. In: Pattern Recognition and Image Analysis. SMPAI, vol. 1, pp. 49–61 (1992). https://doi.org/10.1142/9789812797902_0004
139. Pacharoen, W., Aoki, T., Bhattarakosol, P., Surarerks, A.: Active learning of nondeterministic finite state machines. Math. Probl. Eng. **2013** (2013)
140. Peled, D.A., Vardi, M.Y., Yannakakis, M.: Black box checking. J. Autom. Lang. Comb. **7**(2), 225–246 (2002). https://doi.org/10.25596/JALC-2002-225
141. Petrenko, A., Avellaneda, F.: Learning and adaptive testing of nondeterministic state machines. In: QRS, pp. 362–373. IEEE (2019). https://doi.org/10.1109/QRS.2019.00053
142. Petrenko, A., Yevtushenko, N.: Adaptive testing of nondeterministic systems with FSM. In: HASE, pp. 224–228. IEEE Computer Society (2014). https://doi.org/10.1109/HASE.2014.39
143. Pferscher, A., Aichernig, B.K.: Fingerprinting bluetooth low energy devices via active automata learning. In: FM. Lecture Notes in Computer Science, vol. 13047, pp. 524–542. Springer, Berlin (2021). https://doi.org/10.1007/978-3-030-90870-6_28
144. Pferscher, A., Wunderling, B., Aichernig, B.K., Muskardin, E.: Mining digital twins of a VPN server. In: FMDT@FM. CEUR Workshop Proceedings, vol. 3507. CEUR-WS.org (2023)
145. Quinlan, J.R.: The effect of noise on concept learning. In: Machine Learning, an Artificial Intelligence Approach Volume II, Chap. 6, pp. 149–166. Morgan Kaufmann, San Mateo (1986)

146. Raffelt, H., Steffen, B., Berg, T.: LearnLib: a library for automata learning and experimentation. In: FMICS, pp. 62–71. ACM, New York (2005). https://doi.org/10.1145/1081180.1081189
147. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Inf. Comput. **103**(2), 299–347 (1993). https://doi.org/10.1006/INCO.1993.1021
148. Sabahi-Kaviani, Z., Ghassemi, F., Alimadadi, Z.: Combining machine and automata learning for network traffic classification. In: TTCS. Lecture Notes in Computer Science, vol. 12281, pp. 17–31. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-57852-7_2
149. Sanchez, L., Groote, J.F., Schiffelers, R.R.H.: Active learning of industrial software with data. In: FSEN. Lecture Notes in Computer Science, vol. 11761, pp. 95–110. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-31517-7_7
150. Schuts, M., Hooman, J., Vaandrager, F.W.: Refactoring of legacy software using model learning and equivalence checking: an industrial experience report. In: IFM. Lecture Notes in Computer Science, vol. 9681, pp. 311–325. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-33693-0_20
151. Schwammberger, M., Klös, V.: From specification models to explanation models: an extraction and refinement process for timed automata. In: FMAS/ASYDE@SEFM. EPTCS, vol. 371, pp. 20–37 (2022). https://doi.org/10.4204/EPTCS.371.2
152. Selvaraj, Y., Farooqui, A., Panahandeh, G., Fabian, M.: Automatically learning formal models: an industrial case from autonomous driving development. In: MoDELS (Companion), pp. 33:1–33:10. ACM, New York (2020). https://doi.org/10.1145/3417990.3421262
153. Selvaraj, Y., Farooqui, A., Panahandeh, G., Ahrendt, W., Fabian, M.: Automatically learning formal models from autonomous driving software. Electronics **11**(4) (2022). https://doi.org/10.3390/electronics11040643
154. Shahbaz, M., Groz, R.: Analysis and testing of black-box component-based systems by inferring partial models. Softw. Test. Verif. Reliab. **24**(4), 253–288 (2014). https://doi.org/10.1002/STVR.1491
155. Simão, T.D., Suilen, M., Jansen, N.: Safe policy improvement for POMDPs via finite-state controllers. In: AAAI, pp. 15109–15117. AAAI Press, Menlo Park (2023). https://doi.org/10.1609/AAAI.V37I12.26763
156. Soucha, M., Bogdanov, K.: Observation tree approach: active learning relying on testing. Comput. J. **63**(9), 1298–1310 (2020). https://doi.org/10.1093/COMJNL/BXZ056
157. Spaan, M.T.J.: Partially observable Markov decision processes. In: Reinforcement Learning. Adaptation, Learning, and Optimization, vol. 12, pp. 387–414. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-27645-3_12
158. Starre, R.A.N., Loog, M., Oliehoek, F.A.: An analysis of abstracted model-based reinforcement learning (2022). CoRR arXiv:2208.14407. https://doi.org/10.48550/ARXIV.2208.14407
159. Tang, X., Shen, W., Zhang, M., An, J., Zhan, B., Zhan, N.: Learning deterministic one-clock timed automata via mutation testing. In: ATVA. Lecture Notes in Computer Science, vol. 13505, pp. 233–248. Springer, Berlin (2022). https://doi.org/10.1007/978-3-031-19992-9_15
160. Tappler, M., Aichernig, B.K.: Differential safety testing of deep RL agents enabled by automata learning. In: AISoLA. Lecture Notes in Computer Science, vol. 14380, pp. 138–159. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-46002-9_8
161. Tappler, M., Aichernig, B.K., Bloem, R.: Model-based testing iot communication via active automata learning. In: ICST, pp. 276–287. IEEE Computer Society (2017). https://doi.org/10.1109/ICST.2017.32
162. Tappler, M., Aichernig, B.K., Bacci, G., Eichlseder, M., Larsen, K.G.: L^*-based learning of Markov decision processes (extended version). Form. Asp. Comput. **33**(4–5), 575–615 (2021). https://doi.org/10.1007/S00165-021-00536-5
163. Tappler, M., Pranger, S., Könighofer, B., Muškardin, E., Bloem, R., Larsen, K.G.: Automata learning meets shielding. In: ISoLA (1). Lecture Notes in Computer Science, vol. 13701, pp. 335–359. Springer, Berlin (2022). https://doi.org/10.1007/978-3-031-19849-6s_20
164. Tappler, M., Muskardin, E., Aichernig, B.K., Könighofer, B.: Learning environment models with continuous stochastic dynamics (2023). CoRR arXiv:2306.17204. https://doi.org/10.48550/ARXIV.2306.17204
165. Tavassoli, S., Damasceno, C.D.N., Khosravi, R., Mousavi, M.R.: Adaptive behavioral model learning for software product lines. In: SPLC (A), pp. 142–153. ACM, New York (2022). https://doi.org/10.1145/3546932.3546991
166. Tonella, P., Marchetto, A., Nguyen, D.C., Jia, Y., Lakhotia, K., Harman, M.: Finding the optimal balance between over and under approximation of models inferred from execution logs. In: ICST, pp. 21–30. IEEE Computer Society (2012). https://doi.org/10.1109/ICST.2012.82
167. UPPAAL: https://uppaal.org/
168. Urbat, H., Schröder, L.: Automata learning: an algebraic approach. In: LICS, pp. 900–914. ACM, New York (2020). https://doi.org/10.1145/3373718.3394775
169. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. **22**(5), 297–312 (2012). https://doi.org/10.1002/STVR.456
170. Vaandrager, F.W.: Model learning. Commun. ACM **60**(2), 86–95 (2017). https://doi.org/10.1145/2967606
171. Vaandrager, F.W., Garhewal, B., Rot, J., Wißmann, T.: A new approach for active automata learning based on apartness. In: TACAS (1). Lecture Notes in Computer Science, vol. 13243, pp. 223–243. Springer, Berlin (2022). https://doi.org/10.1007/978-3-030-99524-9_12
172. Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984). https://doi.org/10.1145/1968.1972
173. Valmari, A.: The state explosion problem. In: Petri Nets. Lecture Notes in Computer Science, vol. 1491, pp. 429–528. Springer, Berlin (1996). https://doi.org/10.1007/3-540-65306-6_21
174. van den Bos, P.: Coverage and games in model-based testing. Ph.D. thesis, Radboud University (2020)
175. van den Bos, P., Smetsers, R., Vaandrager, F.W.: Enhancing automata learning by log-based metrics. In: IFM. Lecture Notes in Computer Science, vol. 9681, pp. 295–310. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-33693-0_19
176. van Heerdt, G., Sammartino, M., Silva, A.: CALF: categorical automata learning framework. In: CSL. LIPIcs, vol. 82, pp. 29:1–29:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017). https://doi.org/10.4230/LIPICS.CSL.2017.29
177. van Heerdt, G., Kupke, C., Rot, J., Silva, A.: Learning weighted automata over principal ideal domains. In: FoSSaCS. Lecture Notes in Computer Science, vol. 12077, pp. 602–621. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-45231-5_31
178. van Heerdt, G., Sammartino, M., Silva, A.: Learning automata with side-effects. In: CMCS. Lecture Notes in Computer Science, vol. 12094, pp. 68–89. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-57201-3_5
179. Verwer, S., Eyraud, R., de la Higuera, C.: Results of the PAutomaC probabilistic automaton learning competition. In: ICGI. JMLR Proceedings, vol. 21, pp. 243–248. JMLR.org (2012)
180. Vila, P., Ganty, P., Guarnieri, M., Köpf, B.: CacheQuery: learning replacement policies from hardware caches. In: PLDI, pp. 519–532. ACM, New York (2020). https://doi.org/10.1145/3385412.3386008

181. Volpato, M., Tretmans, J.: Approximate active learning of nondeterministic input output transition systems. Electron. Commun. EASST **72** (2015). https://doi.org/10.14279/TUJ.ECEASST.72.1008
182. Waga, M.: Active learning of deterministic timed automata with Myhill-Nerode style characterization. In: CAV (1). Lecture Notes in Computer Science, vol. 13964, pp. 3–26. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-37706-8_1
183. Walkinshaw, N., Bogdanov, K., Derrick, J., París, J.: Increasing functional coverage by inductive testing: a case study. In: ICTSS. Lecture Notes in Computer Science, vol. 6435, pp. 126–141. Springer, Berlin (2010)
184. Weiss, G., Goldberg, Y., Yahav, E.: Extracting automata from recurrent neural networks using queries and counterexamples (extended version). Mach. Learn. **113**(5), 2877–2919 (2024). https://doi.org/10.1007/S10994-022-06163-2
185. Windmüller, S., Neubauer, J., Steffen, B., Howar, F., Bauer, O.: Active continuous quality control. In: CBSE, pp. 111–120. ACM, New York (2013). https://doi.org/10.1145/2465449.2465469
186. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.S.: Formal methods: practice and experience. ACM Comput. Surv. **41**(4), 19:1–19:36 (2009). https://doi.org/10.1145/1592434.1592436
187. Xu, R., An, J., Zhan, B.: Active learning of one-clock timed automata using constraint solving. In: ATVA. Lecture Notes in Computer Science, vol. 13505, pp. 249–265. Springer, Berlin (2022). https://doi.org/10.1007/978-3-031-19992-9_16
188. Yang, N., Aslam, K., Schiffelers, R.R.H., Lensink, L., Hendriks, D., Cleophas, L., Serebrenik, A.: Improving model inference in industry by combining active and passive learning. In: SANER, pp. 253–263. IEEE (2019). https://doi.org/10.1109/SANER.2019.8668007