Active Automata Learning in Practice
An Annotated Bibliography of the Years 2011 to 2016

Falk Howar[1(✉)] and Bernhard Steffen[2]

[1] Dortmund University of Technology and Fraunhofer ISST, Dortmund, Germany
falk.howar@tu-dortmund.de
[2] Dortmund University of Technology, Dortmund, Germany
steffen@cs.tu-dortmund.de

Abstract. Active automata learning is slowly becoming a standard tool in the toolbox of the software engineer. As systems become ever more complex and development becomes more distributed, inferred models of system behavior become an increasingly valuable asset for understanding and analyzing a system's behavior. Five years ago (in 2011) we have surveyed the then current state of active automata learning research and applications of active automata learning in practice. We predicted four major topics to be addressed in the then near future: efficiency, expressivity of models, bridging the semantic gap between formal languages and analyzed components, and solutions to the inherent problem of incompleteness of active learning in black-box scenarios. In this paper we review the progress that has been made over the past five years, assess the status of active automata learning techniques with respect to applications in the field of software engineering, and present an updated agenda for future research.

1 Introduction

Active automata learning [13] infers models from observations. Alternating between deriving conjectures from experiments and observations and then trying to corroborate or disprove conjectures, active automata learning can be seen as one instance of the fundamental method in science described by Popper [126], who postulates that models (of the world) cannot be verified or reliably generalized, not even probabilistically. Rather, models (of the world) can only be falsified and repaired.

In a world of evermore complex man-made systems that consist of many components, developed by large groups of engineers, and use independently developed libraries or basic software, referring to Popper's scientific method seems appropriate for analyzing, understanding, and validating the behavior of (software) systems whose complexity is beyond the reach of deductive methods.

When learning the behavior of software systems, an observation can be a simple execution of some target component, or a sequence of packages exchanged with a networked system, but also an instance of model checking the feasibility

A. Bennaceur et al. (Eds.): ML for Dynamic Software Analysis, LNCS 11026, pp. 123–148, 2018.
https://doi.org/10.1007/978-3-319-96562-8_5

of a sequence of steps on a system. Learned models enable the application of (formal) analysis and verification techniques or testing approaches, e.g., of model checking [42] or model-based testing [27].

This has first been demonstrated in the concrete scenario of testing computer telephony integrated (CTI) systems [12,69,71], where a finite automaton model was inferred from experiments and used as a basis for regression testing. When analyzing bigger systems, however, it became clear quickly that the success of automata learning in practice would hinge on practical optimizations like efficient implementation of existing learning algorithms, strategies for selecting experiments, and the development of new and more efficient learning algorithms that infer more expressive models. An example of early practical optimizations is the exploitation of the prefix-closedness of a system's set of traces for generating observations to many experiments performed by a learning algorithm without executing actual tests on a system [72,86,107,140].

In 2011, we wrote a report and published a book chapter on the practical challenges in applying active automata learning in software engineering and surveyed the then current state of active automata learning research and applications [81]. We predicted four major topics to be addressed in the then near future: efficiency, expressivity of models, bridging the semantic gap between formal languages and analyzed components, and solutions to the inherent problem of incompleteness of active learning in black-box scenarios. In this paper, we survey the literature on active automata and provide a brief overview of the progress that has been made in the years 2011 to 2016 towards these challenges.

Organization. We give a very brief introduction to active automata learning in Sect. 2 and revisit the challenges that we identified in 2011 in Sect. 3. Section 4 surveys work that focuses on active automata learning in software engineering in the past six years (i.e., from 2011 to 2016). Finally, we asses the progress that has been made and update the list of challenges, taking into account the results of the survey, in Sect. 5.

2 Active Automata Learning

Active automata learning [13] is concerned with the problem of inferring an automaton model for an unknown formal language L over some alphabet Σ.

MAT Model. Active learning is often formulated as a cooperative game between a learner and a teacher, as is sketched in Fig. 1. The task of the learner is to learn a model of some unknown formal language L. The teacher can assist the learner by answering two kinds of queries:

Membership queries ask for a single word $w \in \Sigma^*$ if it is in the unknown language L. The teacher answers these queries with "yes" or "no".

Equivalence queries ask for a candidate language L_H if L_H equals L. In case a conjectured language L_H does not equal L, the teacher will provide a counterexample: a word from the symmetric difference of L_H and L.

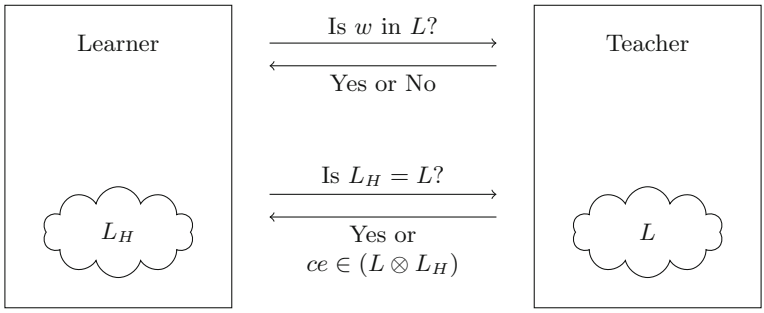

Fig. 1. Active automata learning in the MAT model.

The teacher in this model is called a *minimally adequate teacher* (MAT) and the learning model is hence often referred to as MAT learning.

Dana Angluin's original contribution (in hindsight) is twofold: with the MAT learning model, she introduced an abstraction that allowed for the separation of concerns (constructing stable preliminary models and checking the correctness of these models). This enabled an algorithmic pattern or framework of reasoning that allowed the formulation and optimization of learning algorithms. The second contribution is the original L^* learning algorithm for regular languages and a sequence of lemmas on the status of preliminary models, e.g., showing that for L^* all conjectured models are consistent with all previous observations. The learning algorithm and the sequence of lemmas have served as a basis for proving corresponding properties for many learning algorithms for more complex classes of concepts.

Languages and Automata. A conjectured language L_H is represented by its canonical deterministic acceptor and identified using its residual languages. Intuitively, a residual language [48] of a language is the language after some prefix. Formally, for some language L and a word $u \in \Sigma^*$, the residual language $u^{-1}L$ is the set $\{w \in \Sigma^* \mid uw \in L\}$. A regular language L can be characterized by a finite set of residual languages and every state of the language's canonical acceptor corresponds to one of these languages.

Definition 1. *A Deterministic Finite Automaton (DFA) is a tuple* $A = \langle Q, q_0, \Sigma, \delta, F \rangle$ *where:*

- Q *is a finite nonempty set of states,*
- $q_0 \in Q$ *is the initial state,*
- Σ *is a finite alphabet,*
- $\delta : Q \times \Sigma \to Q$ *is the transition function, and*
- $F \subseteq Q$ *is the set of accepting states.*

We extend δ *to words in the natural way by defining* $\delta(q, \epsilon) = q$ *for the empty word* ϵ *and* $\delta(q, ua) = \delta(\delta(q, u), a)$ *for* $u \in \Sigma^*$ *and* $a \in \Sigma$*. A word* w *is accepted by* A *if* $d(q_0, w) \in F$*.*

The well-known Nerode congruence is the basis for the construction of the canonical acceptor for a regular language L: Two words u, v are Nerode-equivalent w.r.t. L if their residual languages in L are identical [120]. The canonical DFA A_L for L has one state for every residual language of L (i.e., for every class of the Nerode-relation induced by L). The residual language $u^{-1}L$ after word u is represented by state q_u. With $q_\epsilon = q_0$ and $\delta(q_u, a) = q_{ua}$ for $u \in \Sigma^*$ and $a \in \Sigma$ in A_L, a word u leads to the state $q_u = \delta(q_0, u)$, representing the corresponding residual language $u^{-1}L$. Finally, letting $q_u \in F$ iff $u \in L$ makes A_L an acceptor for L and A_L^u, the automaton obtained from A_L by making q_u the initial state, an acceptor for $u^{-1}L$.

The L* Algorithm. Active learning algorithms are based on the dual characterization of states in the canonical acceptor A_L, by words leading to states and their residual languages. The key observation is that words u and u' cannot lead to the same state if for some $v \in \Sigma^*$ the word uv is in L while $u'v$ is not in L (or vice versa). Thus, a finite nonempty set U of *prefixes* can be used to identify states and a finite nonempty set V of *suffixes* can be used to distinguish states.

The L^* algorithm[1] for regular languages uses an observation table $Obs : (U \cup U \cdot \Sigma) \times V \mapsto \{1, 0\}$ for organizing results of membership queries, letting $Obs(u, v) = 1$ iff $uv \in L$ and 0 otherwise. Sets U and V are initialized as $\{\epsilon\}$, i.e., with a prefix for the initial state and a suffix that distinguishes final states from non-final states, and are extended incrementally. An automaton A_{Obs} can be generated from Obs with states q_u for $u \in U$, initial state q_ϵ, transitions $\delta(q_u, a) = q_{u'}$ for $u, u' \in U$ and $a \in \Sigma$ where $Obs(ua) = Obs(u')$ and $q_u \in F$ iff $Obs(u, \epsilon) = 1$. This automaton is only well-defined if δ is total. The algorithm ensures this when extending the table (and hence refining the corresponding automaton) on the basis of query results, iterating the two main steps (1) establishing *closedness* via membership queries, and (2) *testing for equivalence* via equivalence queries.

Local exploration. The first phase checks whether the knowledge, gathered from membership queries and accumulated in the observation table, suffices to construct a hypothesis automaton with a total transition function. This requires that the table is *closed*, meaning that for every word $w \in U \cdot \Sigma$ there is a prefix $u \in U$ with $Obs(u) = Obs(w)$. The set U of prefixes is extended by words from $U \cdot \Sigma$ until the table is closed and a hypothesis automaton A_{Obs} (accepting L_H in Fig. 1) can be generated.

Checking Equivalence. An equivalence query checks whether A_{Obs} is the canonic acceptor of the target language L. Once this is true, the learning procedure terminates successfully. Otherwise, the equivalence query returns a counterexample from the symmetric difference of L and L_H. As was shown by Rivest and

[1] We only provide a very brief sketch of the improved version of L^* due to Rivest and Schapire here [130]. A more detailed presentation can be found in Angluin's original paper [13].

Schapire [130], a counterexample w indicates that the set V of suffixes, approximating the characterization of states by residual languages, can be refined by adding one of the suffixes of w to V: The word $w = a_1 \cdots a_m$ traverses states $q_0, q_1, \ldots, q_m$ in A_{Obs}. For index $0 \leq i \leq m$, let $\tilde{u}_i$ be the prefix for q_i in U with $\delta(q_0, \tilde{u}_i) = q_i$ and for $1 \leq j \leq m$ let v_j be the suffix $a_j \cdots a_m$ of w. There is a pair $\tilde{u}_{i-1}$ and $\tilde{u}_i$ of prefixes in U with $Obs(\tilde{u}_{i-1}a_i) = Obs(\tilde{u}_i)$ while $\tilde{u}_{i-1}a_i \cdot v_{i+1} \in L$ and $\tilde{u}_i \cdot v_{i+1} \notin L$ (or vice versa).[2] Adding v_{i+1} to the set V of suffixes will lead to unclosedness of the observation table, which in turn will lead to adding prefixes to U, and result in a refined conjecture.

Correctness and Termination. The correctness argument for this approach follows a straightforward pattern, which does not only hold for L^*, but also for all of the derivatives [96,107,123,130,132] presented so far.

Partial correctness [75] is obvious, because learning only terminates after the equivalence oracle guaranteed the correctness of the inferred model. What remains to be shown is termination. The following four steps suffice to prove that the learning procedure always terminates after at most n equivalence queries, where n is the number of states of the desired minimal acceptor for L:

1. The state construction, using distinguishing suffixes in lieu of residual languages, guarantees that the number of states of the hypothesis automaton can never exceed the number of states of the smallest deterministic automaton accepting the considered language.
2. The closedness procedure guarantees that each transition of the hypothesis automaton is represented by a prefix during learning. This means in particular that a hypothesis automaton of the size (in terms of number of states) of the smallest deterministic automaton for the considered language must already be isomorphic to this (canonic) automaton.
3. The analysis of counterexamples guarantees that at least one additional state is added to the hypothesis automaton for each counterexample. Thus, due to (1) and (2), such treatments can happen only n times.
4. The equivalence checking mechanism, often called equivalence oracle, provides new counterexamples as long as the language of the hypothesis automaton does not match the desired result.

Using the underlying concept of query learning a number of optimizations and akin algorithms have been proposed in the 1990s [96,130], Balcázar et al. give a unifying overview [17].

Application in Model Learning. To meet the requirements in practical scenarios, Margaria et al. transferred automata learning to Mealy machines [107].

[2] While the word $w = \tilde{u}_0 \cdot v_1$ is a counterexample, $\tilde{u}_m$ cannot be a counterexample (by construction of A_{Obs}), and all words $\tilde{u}_i \cdot v_{i+1}$ with $0 \leq i \leq m-1$ lead to the same state as $\tilde{u}_m$ in A_{Obs}. As a consequence, at one index $\tilde{u}_{i-1}a_i \cdot v_{i+1} \in L$ and $\tilde{u}_i \cdot v_{i+1} \notin L$ (or vice versa).

Mealy machines are widely used models of deterministic reactive systems and multiple optimized algorithms have been proposed [72,129,132]. Examples of applications in the years before 2011 are the learning of behavioral models for Web Services [127], communication protocol entities [139], or software components [86,117,128].

Extensions to inference methods focus on modeling phenomena that occur in real systems. On the basis of inference algorithms for Mealy machines, inference algorithms for I/O-automata [8], timed automata [124], Petri Nets [52], and Message Sequence Charts [22,23] have been developed. With the I/O-automata model, a wide range of systems that comprise quiescence is made accessible for query learning. Timed automata model explicitly time dependent behavior. With Petri Nets, systems with explicit parallel state are addressed.

3 A Short Review of Challenges in Applications

In this section, we discuss the challenges we identified in 2011 for the practical application of active automata learning. Automata learning can be considered as a key technology for dealing with *black-box systems*, i.e., systems that can be observed, but for which no or little knowledge about the internal structure is available. *Active* automata learning is characterized by its specific means of observation, i.e., its proactive way of posing membership queries and equivalence queries. It requires some way to realize this query-based interaction for the considered application contexts. Whereas membership queries may often be realized via testing in practice, equivalence queries are typically unrealistic.

3.1 Interacting with Real Systems

The interaction with a realistic target system comes with a number of challenges. A merely technical problem is establishing an adequate interface that allows one to realize membership queries. This can be rather simple for systems designed for connectivity (e.g., Web-services or libraries) which have a native concept of being invoked from the outside and come with documentation on how to accomplish this (cf. the work on so-called dynamic Web testing [128]). It may be more difficult for other systems, e.g., embedded systems that work on streams of data.

Establishing an adequate abstraction for learning is a second challenge: An abstraction has to produce a useful and finite model while at the same time allowing for an automatic back and forth translation between the abstract model and the concrete target system. At the time, there was some work focusing explicitly on the use of abstraction in learning [4,6] and even first steps in the direction of automatic abstraction refinement [84,89].

Another challenge is that active learning requires membership queries to be independent. Solutions range here from reset mechanisms via homing sequences [130] or snapshots of the system state to the generation of observably equivalent initial conditions. E.g., for session-based protocols, it may be sufficient to perform every membership query with a fresh session identifier [81].

3.2 Efficiency

Whereas small learning experiments typically require only a few hundred membership queries, learning realistic systems may easily require several orders of magnitude more. In some scenarios, each membership query may need multiple seconds or sometime even minutes to compute. In such a case minimizing the number of required membership queries is the key to success.

In [83,132] optimizations are discussed to classic learning algorithms that aim at saving membership queries in practical scenarios. Additionally, the use of filters (exploiting domain specific expert knowledge) has been proven as a practical solution to the problem [72,140]. Finally, the choice of a concrete learning algorithm may have a huge influence on the number of membership queries that are used to infer a model of a target system [77].

3.3 Expressivity of Models

Active learning classically is based on abstract communication alphabets. Parameters and interpreted values are only treated to an extend expressible within the abstract alphabet. In practice, this typically is not sufficient, not even for systems as simple as communication protocols, where, e.g., increasing sequence numbers must be handled, or where authentication requires matching user/password combinations.

First attempts to deal with parameters in models range from case studies with manual solutions [117] to extensions of learning algorithms that can deal with Boolean parameters [137,138]. One big future challenge at the time was extending learning to models with state variables and arbitrary data parameters in a more generic way, as explored by [10].

3.4 Equivalence Queries

Equivalence queries compare a learned hypothesis model with the target system for language equivalence and, in case of failure, return a counterexample exposing a difference. Their realization is rather simple in white-box scenarios: equivalence can be checked. In black-box scenarios, however, equivalence queries have to be approximated using membership queries. Without the introduction of additional assumptions, such equivalence tests are not decidable: the possibility of having not tested extensively enough always remains.

Conformance testing has been used to simulate equivalence queries. If, e.g., an upper bound on the number of states the target system can have is known, the W-method [41] or the Wp-method [57] can be applied. Both methods have an exponential complexity (in the size of the target system and measured in the number of membership queries needed). The relationship between regular extrapolation and conformance testing methods is discussed in [19].

Without introducing any additional assumptions, only approximate solutions exploiting membership queries exist. Here, conformance testing methods may not always be a wise choice. It has turned out that changing the view from

"trying to proof equivalence", e.g., by using conformance testing techniques, to "finding counter examples fast" has a strong positive impact. An attempt to intensify research in this direction was the 2010 Zulu challenge [44]. The winning solution is discussed in [83]. The main contribution of this solution is a strategy for sharing information on test coverage for the evolving model between individual equivalence queries.

4 Recent Advances in Active Automata Learning

In order to assess the advances in practical application of active automata learning in the domain of Software engineering, we survey the literature on active automata learning in the years 2011 to 2016. Basis for the survey are the ACM's digital library and the proceedings of several big software engineering conferences (i.e., ICSE, CAV, ETAPS). The survey is not exhaustive since the number of candidate publications is in the thousands. We have used different heuristics for filtering out relevant publications: cited foundational papers, used keywords, and authors known to work in the field.

We sort publications into categories based on their main focus and differentiate between advances that have been made by application of automata learning and those that have been made to the methodology of active automata learning itself. Finally we discuss advances in lines of work that are closely related (i.e., learning from examples or active learning of other classes of concepts).

4.1 Advances in Applications

There have been impressive advances in the application of automata learning in diverse scenarios over the past years. Applications are found in black-box contexts as well as in white-box scenarios. The broad range of application areas documents that active (automata) learning is becoming one of the well-established tools in the toolbox of the formal methods trained software engineer.

Specification Generation. The most obvious application of active automata learning is the *a posteriori* generation of specifications from prototypes or from running (legacy) systems. Esparza et al. present a learning algorithm for Workflow Petri Nets [52] using log data and a teacher that answers executability of conjectured workflows. Sun et al. use active automata learning in combination with automated abstraction refinement and random testing for finding abstract behavioral models of Java classes [136]. Aarts et al. demonstrate how a combination of active automata learning with manually crafted abstraction mappers can be used to infer models of the SIP and TCP protocols [7]. Gu and Roychowdhury present a variant of L^* for inferring finite state abstractions of continuous circuits defined by differential equations [67]. Aadithya and Roychowdhury go on and extend the approach from learning regular abstractions to models with arbitrary I/O alphabets [1].

Model-based Testing without Models. One of the earliest practical applications of active automata learning was testing of telecommunication systems [12,69]. The idea of model-based testing without (a priori) models was later elaborated and, e.g., applied to Web-based systems [128]. In recent years this line of application has been continued for several different types of systems. Dinca et al. develop an approach for generating test-suites for Event-B models through active automata learning [49,50]. Choi et al. use active automata learning for testing the behavior of the graphical user interfaces of Android applications [39,40]. Shahbaz and Groz use automata learning for integration testing [133]. They infer models of embedded components and use these models as a basis for test case generation. Meinke and Sindhu present LBTest, a tool for learning-based testing for reactive systems, integrating model checking, active automata learning, and random testing [111]. In this volume, the relation between learning and testing is discussed in [11] and an overview of learning-based testing is presented in [109].

Software Re-engineering. Inferred models cannot only be used for testing but also for comparing different versions or implementations of a system. This can, e.g., be useful for searching (accidental) differences in the behavior of subsequent version of a software system. Neubauer et al. develop 'active continuous quality control': they use active automata learning on subsequent versions of a Web-application (during development) and analyze models for unintended behavioral changes between versions [121,122,145]. The approach integrates active automata learning, model checking, regression testing, and risk-based testing. Schuts et al. use model learning and equivalence checking to assist re-engineering of legacy software in an industrial context at Philips [131]. Howar et al. validate a model-to-code translator. They use active automata learning to extract behavioral models from generated implementations and compare these models to specification models [80]. Bainczyk et al. presented an easy to use tool for mixed-mode learning, which, in particular, allows one to compare back-end and front-end functionality of web applications [16].

Verification and Validation. Inferred models can be used in (formal) verification and system analysis as well, as sketched below.

Security. Behavioral models of systems can be used for identifying vulnerabilities. Active learning in these scenarios is often used to automate the work of a prospective attacker, exploring state of a systems in a structured way. Over the past years, a number of real vulnerabilities have been identified in this manner.

Cho et al. present MACE an approach for concolic exploration of protocol behavior. The approach uses active automata learning for discovering so-called deep states in the protocol behavior. From these states, concolic execution is employed in order to discover vulnerabilities [38]. Chalupar et al. use active automata learning and a LEGO robot to physically interact with smart cards and reverse engineer their protocols [34]. De Ruiter and Poll use active automata

learning for inferring models of TLS implementations and discover previously unknown security flaws in the inferred models [46].

Botinčan and Babić present a learning algorithm for inferring models of stream transducers that integrates active automata learning with symbolic execution and counterexample-guided abstraction refinement [26]. They show how the models can be used to verify properties of input sanitizers in Web-applications. Xue et al. use active automata learning for inferring models of JavaScript malware [146].

Argyros et al. present SFADiff, a tool based on active automata learning for inferring symbolic automata that characterize the difference between similar programs [15]. The work is motivated by the security challenge of fingerprinting programs based on their behavior.

Safety/Correctness. Active learning can be used to generate (abstract) models at interfaces of systems. The behavior at such interfaces is often an issue when integrating systems into environments (virtual or physical). Inferred models can be used for analyzing safety in such situations. Combéfis et al. use active learning to generate abstract models of systems as a basis for analyzing potential mode confusion, a well-known problem in human machine interaction [45], while Howar et al. use the register automaton model [32] for inferring precise semantic interfaces of data structures [79]. Giannakopoulou et al. develop an active learning algorithm that infers safe interfaces of software components with guarded actions. In their model, the teacher is implemented using concolic execution [62,78]. Khalili et al. [97] use active automata learning to obtain behavioral models of the middleware of a robotic platform. The models are used during verification of control software for this platform. Fiterău-Broştean et al. use learning and model checking to analyze the behavior of different implementations of the TCP protocol stack and document several instances of implementations violating RFC specifications [56].

Assume-Guarantee Reasoning. Assume-guarantee reasoning has been a big area of application of active automata learning algorithms for much longer than the past couple of years (cf. [43,101,125]). The moderate style of exploration that is achieved by learning is used to reduce the problem of state space explosion. Recent advances have been made by finding active automata learning to many classes of systems. Learning algorithms are usually based quite directly on the classic L^* algorithm. The required extensions in expressivity of models are usually realized through powerful teachers.

Chaki and Gurfinkel infer assumptions for ω-regular systems [33]. He et al. present a framework automated symbolic assume-guarantee reasoning that incorporates a MAT learning algorithm for BDDs [74]. He and some of the same and some other co-authors also present a compositional reasoning framework for concurrent probabilistic systems using an active learning algorithm for multi-terminal binary decision diagrams [73]. Feng et al. present an algorithm for inferring assumptions for probabilistic assume/guarantee reasoning [53,54]. Komuravelli et al. develop automata learning of non-deterministic probabilistic models that

then serve as assumptions during automated assume/guarantee reasoning [99]. Meller et al. develop learning-based assume-guarantee reasoning for behavioral UML systems, using the L^* algorithm "off the shelf" [112].

Synthesis. The latest area of application of active automata learning that could be identified is synthesis. In synthesis, active learning is used for exploring and constructing formal models of safe (emerging) behavior that can be used as a basis for synthesizing safe mediators or controllers. Lin and Hsiung use learning-based assume-guarantee reasoning to build a compositional synthesis algorithm [104]. Cheng et al. synthesize safe and deadlock-free component-based systems using priorities and automated assumption learning [37]. Neider and Topcu use active automata learning to solve safety games [118,119].

4.2 Tools and Libraries

There are many tools presented in the work surveyed that integrate active learning algorithms. For this category we focus on tools and libraries that provide active automata learning algorithms to applications.

Since 2004, Bernhard Steffen's group develops LearnLib[3], a library for active automata learning that comprises infrastructure (e.g., filters and abstractions) for learning models of real-world systems [116,129]. Merten et al. present an extension of LearnLib for inferring data-aware models of Web-Services [115] fully automatically using only WSDL interface descriptions to bootstrap the learning process. The maturity of today's version of the LearnLib, which is now open source, is witnessed by the CAV 2015 artifact award [91].

There exist at least two other open-source automata learning libraries that provide implementations of textbook algorithms, complemented by own developments: libalf[4], the *Automata Learning Framework* [24], was developed primarily at the RWTH Aachen. Its active development seems to have ceased; the last version was released in April 2011. AIDE[5] [98], the *Automata-Identification Engine*, developed by a group at University of Genoa. It is not clear from the web-page if the library is still maintained.

Several tools and libraries for learning more expressive automata models have been developed over the past couple of years. The *Tomte*[6] [3] tool is developed at Radboud University. The tool fully automatically constructs abstractions (i.e., mappers) for automata learning and uses LearnLib for inferring models. Drews and D'Antoni develop a library for symbolic automata and symbolic visibly pushdown automata[7] [51]. The library provides learning algorithms for symbolic automata. Cassel et al. develop *RaLib*[8] [134], an extension to LearnLib for

[3] https://learnlib.de/.
[4] http://libalf.informatik.rwth-aachen.de/.
[5] http://aide.codeplex.com/.
[6] http://tomte.cs.ru.nl/.
[7] https://github.com/lorisdanto/symbolicautomata.
[8] https://bitbucket.org/learnlib/ralib/.

learning algorithms that infer extended finite state machine models. All three tools seem to be actively maintained.

4.3 Algorithmic Advances

While active automata learning has gained a lot of traction as a tool in software engineering applications, there is another group of work aiming at improving the foundations of active automata learning by extending it to semantically richer models, by developing more efficient learning algorithms, by exploring new learning models, and by working on techniques for approximating equivalence queries in black-box scenarios that yield quantifiable correctness guarantees for inferred models.

Expressivity. Meinke and Sindhu present IKL, a learning algorithm in the MAT model that infers Kripke structures [110]. Lin et al. develop a mixed active and passive learning algorithm that infers a subclass of timed automata, so-called event-recording automata [103].

Howar et al. extend active automata learning in the MAT model to register automata, which model control-flow as well as data-flow between data parameters of inputs, outputs, and a set of registers [82,114]. Registers and data parameters can be compared for equality. The authors demonstrate the effectiveness of their approach by inferring models of data structures [79] and extent the expressivity to allow for arbitrary data relations that meet certain learnability criteria [31,134]. A summary of this work can be found in this volume [30]. Aarts et al. develop a slightly different approach for inferring register automata models that can compare registers and data parameter for equality [2,3]. The two approaches are compared in [5].

Garg et al. develop an active learning algorithm for so-called quantified data automata over words that can model quantified invariants over linear data structures [59]. Volpato and Tretmans investigate the necessary assumptions under which models of nondeterministic systems can be inferred [141]. Kasprzik shows how residual finite-state tree automata can be inferred from membership queries and positive examples [95]. Isberner presents an active learning algorithm that infers visibly push-down automata [88].

Learning Models. Abel and Reinecke address the problem of inferring a model of a component that can only be addressed through a given and known intermediate component [9]. Decker et al. present active learning of networks of automata that consist of one base automaton and a number of identical components [47].

Groz et al. present a learning algorithm of scenarios in which the system cannot be reset into a well-defined initial state [66] (an extended version can be found in this volume [65]). Leucker and Neider present an active learning algorithm that learns models from an ‘inexperienced’ teacher, i.e., a teacher that fails to answer some membership queries [102].

A separate line of work focuses on learning regular languages from so-called automatic classes in different learning models [28,29,92,93].

Efficiency. For the case of finite regular languages, Ipate presents an active learning algorithm that infers deterministic finite cover automata and in some cases leads to substantial savings compared to more generic active learning algorithms [87]. Groz et al. develop optimizations of the L^* algorithm targeting large input sets, parameterized inputs, and processing counterexamples [64]. Björklund et al. develop a MAT-model learning algorithm that infers universal automata as a representation of regular languages [21]. Angluin et al. develop learning algorithms for universal automata, and alternating automata [14] and evaluate the performance trade offs for inferring these automata models—compared to deterministic finite state automata.

Means and Maler present a variant of L^* that learns concise models for systems with big sets of inputs by inferring symbolic characterizations of equivalent sets of inputs [113], an approach reminiscent of [89].

Finally, Isberner et al. develop the TTT algorithm [90], a space-optimal active learning algorithm that computes minimal distinguishing suffixes from counterexamples. The TTT algorithms is particularly well-suited when aiming at lifelong learning [20], where equivalence queries are essentially replaced by contínuous monitoring of the running system.

Quality of Models. Van den Bos et al. develop a quality metric for inferred models and introduce a so-called 'Comparator' that can be used to enforce that the quality of intermediate models obtained during learning always increases (w.r.t. to the introduced metric) [25]. Chen et al. use the PAC result presented in Angluin's original paper on L^* for implementing a learning-based framework for program verification [35]. Using a PAC result allows them to quantify the confidence in the verification result in the absence of a perfect equivalence oracle.

4.4 Related Lines of Work

Learning in general is gaining traction in software engineering. This includes active learning of different concepts, as well learning from examples, which is the method of choice when inferring models from logs and traces.

Active Learning of Loop Invariants. One line of work aims at synthesizing invariants for loops in programs using combinations of active learning of logic formulas, predicate abstraction, and counterexample-guided abstraction refinement (e.g., [94]). Konev et al. present an algorithm for learning logic TBoxes from a teacher that answers entailment queries and equivalence queries [100]. Chen and Wang present an active learning algorithm for Boolean functions and use it for inferring loop invariants [36].

Garg et al. develop several learning algorithms for invariants in different learning models. In [60], they present algorithms for inferring Boolean combinations of numerical invariants for scalar variables and for quantified invariants of arrays and dynamic lists. In [61], they infer inductive invariants in a model where the teacher instructs a learner through positive, negative, and implication examples [61].

Learning from Examples. Passive automata learning infers automata models from positive or from positive and negative examples - the learning model is referred to as "in the limit" [63]. Learning from examples has an equally big recent impact in software engineering as active learning. The application scenarios and advances resemble closely the ones of active learning, as the following examples show.

Applications. Walkinshaw uses passively inferred models as a basis for deciding the adequacy of test suites for black-box systems [142]. Adamis et al. mine for sequential patterns (i.e., frequently observed transitions) in conformance test logs and use these to generate a finite state machine model. The finite state model is then used for performance testing [10].

Medhat et al. present an approach for mining hybrid automata specifications from input/output traces using several machine learning techniques [108]. Mao et al. extend the Alergia algorithm that learns probabilistic models from positive examples to more expressive reactive and timed models [106]. They then investigate how these models can be used for model checking and demonstrate the feasibility in a comparison to statistical model checking. Statistical model checking samples the system directly for a property, while in their approach first a model is inferred and then a property is checked on this model.

Another line of work focuses on designing domain-specific languages for Object processing and formatting, e.g., in Excel, and then learning models in the respective DSL from examples [68,85,144]. Barowy et al. learn formatting rules for spread sheet data from examples [18].

Algorithmic Advances. One recent theoretic results is shown by García et al.: the authors prove the existence of polynomial characteristic samples for every order in which states are merged during learning, i.e., sets of examples that allow correct identification of unknown regular languages [58].

Staworko and Wieczorek present learning algorithms that learn XML path queries from positive and negative examples [135]. Walkinshaw et al. develop a passive learning algorithm that infers extended finite state machines that model control-flow and data-flow from [143]. Högberg presents an algorithm for inferring regular tree languages from positive and negative examples [76].

5 Discussion and Open Challenges

The survey of the literature documents progress concerning all challenges that we identified in our earlier work. A careful analysis shows that progress in some directions has been stronger than in other directions. This yields some potential directions for further research.

Interacting with Real Systems. There is, by now, a considerable number of case studies that show how active learning can be beneficial in different scenarios: In our survey, the number of publications that present applications exceeds

the number of publications that focus primarily on algorithmic or theoretic contributions. It can be observed that in black-box scenarios, membership queries are typically realized through tests, and equivalence queries are approximated by tests. In white-box scenarios, both types of queries are often implemented using model-checking or program analysis.

While interaction with real systems is reported in many publications their corresponding conceptual progress is typically small. Specifically, the proposed methods for establishing appropriate abstractions underlying the learning alphabet or for guaranteeing equivalent initial conditions for membership queries are still mostly a case-specific manual effort.

Efficiency and Tools. In the past ten years, many improved active automata learning algorithms have been developed. Some rely on the observation table, the basic data structure introduced by Angluin, and differ from the original L^* algorithm mostly in the way counterexamples are analyzed. Others use decision trees as data structures. Observations clearly indicate the superiority of tree-based algorithms, combined with efficient counter example analysis. It is striking that despite this algorithmic progress, many applications still use the original L^* algorithm or one of the optimized versions that have been developed in the early 1990s. Sometimes heuristics to overcome well-known weaknesses of L^* are even proposed as general achievements.

One future challenge is therefore the systematic transfer of the existing algorithmic improvements into tools. As of today, there seem to exist only very few tools and libraries that are actively maintained and publicly available (compared to, e.g., the tools in the areas of satisfiability modulo theories or automated theorem proving). In these other domains, competitions have been used to encourage development of new methods and implementation of tools. Maybe model learning needs a similar vehicle for driving the transfer of theoretic results into tools.

Expressivity. Over the years, active learning has been extended to produce more expressive models, like register automata, extended finite state machines, visibly push-down automata, event recording automata, or symbolic automata. It appears that most of the corresponding active learning approaches use the L^* algorithm (or one of its variants) as a reference and often adapt correctness proofs (e.g., [51,105]). This does not only result in inefficient solutions, but often also in quite indirect correctness arguments. The more efficient algorithms are typically technically more involved than the original L^* algorithm making their adaptation to new domains harder. On the other hand, e.g., the TTT algorithm reveals very much of the information-theoretic essence of active automata learning which promotes a better understanding and provides significant performance gains.

One direction for future work is therefore leveraging this potential and providing modular conceptual frameworks that support the adaptation of learning algorithms to new domains and classes of models. Conceptual frameworks have to be complemented by implementations enabling the systematic profiling of various learning algorithms in order to identify the best fitting algorithm for a

given application domain. The LearnLib Studio [116] was a first step into this direction and the authors are currently working on transferring this idea to the open-sourced version of LearnLib.

Equivalence Queries. Equivalence queries are mostly addressed on a per-case-study basis. They are usually implemented as conformance tests or through random testing in black-box scenarios. The concrete strategy for generating test cases varies, but there has hardly been progress on efficient and effective methods for approximating equivalence queries. In order to further develop active automata learning to a point where it can be used by verification techniques or for documentation even in industrial (black-box) contexts, equivalence queries will have to provide a quantifiable measure for the likelihood or precision of inferred models.

One way of obtaining such results is the PAC (probably approximately correct) framework. Angluin obtained a PAC result for the original L^* algorithm by implementing equivalence queries using sequences of membership queries [13]. Recently, this result (which had been largely ignored for 20 years) was picked up and extended [55,105]. In some application scenarios "lifelong learning" seems to be an adequate answer, i.e., monitoring running systems relative the current hypothesis model, identifying behavioral discrepancies, and correcting either the model or, if required, the systems. One major obstacle to this approach are the resulting excessively long counterexamples. The TTT algorithm has been specifically designed to address this challenge [90].

6 Conclusions

In the last 15 years, active automata learning, an originally merely theoretical enterprise, has received attention as a method for dealing with black-box or third-party systems in software engineering. Especially, in the past six years (2011 to 2016) active automata learning has found many applications, ranging from security analysis, to testing, to verification, and even synthesis. At the same time, algorithmic and theoretic advances have led to more efficient learning algorithms that can infer more expressive models (e.g. [30], in this volume) Scalability of active automata learning is still a major challenge. Hybrid approaches that complement the power of black box analysis with white box analysis methods seem to emerge as one possible technique for addressing this challenge (cf. [70], in this volume). Summarizing, active automata learning has developed far beyond what could have been anticipated 15 years ago. However, with every solved problem, news questions arise - making active automata learning a very fruitful area of research with increasingly high practical impact.

References

1. Aadithya, K.V., Roychowdhury, J.: DAE2FSM: automatic generation of accurate discrete-time logical abstractions for continuous-time circuit dynamics. In: Proceedings of the 49th Annual Design Automation Conference, DAC 2012, pp. 311–316. ACM, New York (2012)
2. Aarts, F., Fiterau-Brostean, P., Kuppens, H., Vaandrager, F.: Learning register automata with fresh value generation. In: Leucker, M., Rueda, C., Valencia, F.D. (eds.) ICTAC 2015. LNCS, vol. 9399, pp. 165–183. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25150-9_11
3. Aarts, F., Heidarian, F., Kuppens, H., Olsen, P., Vaandrager, F.: Automata learning through counterexample guided abstraction refinement. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 10–27. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32759-9_4
4. Aarts, F., Heidarian, F., Vaandrager, F.: A theory of history dependent abstractions for learning interface automata. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 240–255. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32940-1_18
5. Aarts, F., Howar, F., Kuppens, H., Vaandrager, F.: Algorithms for inferring register automata. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014. LNCS, vol. 8802, pp. 202–219. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45234-9_15
6. Aarts, F., Jonsson, B., Uijen, J.: Generating models of infinite-state communication protocols using regular inference with abstraction. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 188–204. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16573-3_14
7. Aarts, F., Jonsson, B., Uijen, J., Vaandrager, F.: Generating models of infinite-state communication protocols using regular inference with abstraction. Formal Methods Syst. Des. **46**(1), 1–41 (2015)
8. Aarts, F., Vaandrager, F.: Learning I/O automata. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 71–85. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_6
9. Abel, A., Reineke, J.: Gray-box learning of serial compositions of mealy machines. In: Rayadurgam, S., Tkachuk, O. (eds.) NFM 2016. LNCS, vol. 9690, pp. 272–287. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40648-0_21
10. Adamis, G., Kovács, G., Réthy, G.: Generating performance test model from conformance test logs. In: Fischer, J., Scheidgen, M., Schieferdecker, I., Reed, R. (eds.) SDL 2015. LNCS, vol. 9369, pp. 268–284. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24912-4_19
11. Aichernig, B.K., Mostowski, W., Mousavi, M.R., Tappler, M., Taromirad, M.: Model learning and model-based testing. In: Bennaceur, A., Hähnle, R., Meinke, K. (eds.) ML for Dynamic Software Analysis. LNCS, vol. 11026, pp. 74–100. Springer, Cham (2018)
12. Hagerer, A., Hungar, H., Niese, O., Steffen, B.: Model generation by moderated regular extrapolation. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 80–95. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45923-5_6
13. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)

14. Angluin, D., Eisenstat, S., Fisman, D.: Learning regular languages via alternating automata. In: Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI 2015, pp. 3308–3314. AAAI Press (2015)
15. Argyros, G., Stais, I., Jana, S., Keromytis, A.D., Kiayias, A.: SFADiff: automated evasion attacks and fingerprinting using black-box differential automata learning. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 1690–1701. ACM, New York (2016)
16. Bainczyk, A., Schieweck, A., Isberner, M., Margaria, T., Neubauer, J., Steffen, B.: ALEX: mixed-mode learning of web applications at ease. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 655–671. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_51
17. Balcázar, J.L., Díaz, J., Gavaldà, R., Watanabe, O.: Algorithms for learning finite automata from queries: a unified view. In: Du, D.Z., Ko, K.I. (eds.) Advances in Algorithms, Languages, and Complexity, pp. 53–72. Springer, Boston (1997). https://doi.org/10.1007/978-1-4613-3394-4_2
18. Barowy, D.W., Gulwani, S., Hart, T., Zorn, B.: FlashRelate: extracting relational data from semi-structured spreadsheets using examples. SIGPLAN Not. **50**(6), 218–228 (2015)
19. Berg, T., Grinchtein, O., Jonsson, B., Leucker, M., Raffelt, H., Steffen, B.: On the correspondence between conformance testing and regular inference. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 175–189. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31984-9_14
20. Bertolino, A., Calabrò, A., Merten, M., Steffen, B.: Never-stop learning: continuous validation of learned models for evolving systems through monitoring. ERCIM News **2012**(88), 28–29 (2012)
21. Björklund, J., Fernau, H., Kasprzik, A.: Polynomial inference of universal automata from membership and equivalence queries. Inf. Comput. **246**(C), 3–19 (2016)
22. Bollig, B., Katoen, J.-P., Kern, C., Leucker, M.: Replaying play in and play out: synthesis of design models from scenarios by learning. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 435–450. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_33
23. Bollig, B., Katoen, J.-P., Kern, C., Leucker, M.: *Smyle*: a tool for synthesizing distributed models from scenarios by learning. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 162–166. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85361-9_15
24. Bollig, B., Katoen, J.-P., Kern, C., Leucker, M., Neider, D., Piegdon, D.R.: Libalf: the automata learning framework. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 360–364. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_32
25. van den Bos, P., Smetsers, R., Vaandrager, F.: Enhancing automata learning by log-based metrics. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 295–310. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33693-0_19
26. Botinčan, M., Babić, D.: Sigma*: symbolic learning of input-output specifications. In: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2013, pp. 443–456. ACM, New York (2013)
27. Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.): Model-Based Testing of Reactive Systems. LNCS, vol. 3472. Springer, Heidelberg (2005). https://doi.org/10.1007/b137241

28. Case, J., Jain, S., Le, T.D., Ong, Y.S., Semukhin, P., Stephan, F.: Automatic learning of subclasses of pattern languages. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 192–203. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21254-3_14
29. Case, J., Jain, S., Ong, Y.S., Semukhin, P., Stephan, F.: Automatic learners with feedback queries. In: Löwe, B., Normann, D., Soskov, I., Soskova, A. (eds.) CiE 2011. LNCS, vol. 6735, pp. 31–40. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21875-0_4
30. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Extending automata learning to extended finite state machines. In: Bennaceur, A., Hahnle, R., Meinke, K. (eds.) ML for Dynamic Software Analysis. LNCS, vol. 11026, pp. 149–177. Springer, Cham (2018)
31. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Active learning for extended finite state machines. Formal Asp. Comput. **28**(2), 233–263 (2016)
32. Cassel, S., Jonsson, B., Howar, F., Steffen, B.: A succinct canonical register automaton model for data domains with binary relations. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, pp. 57–71. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33386-6_6
33. Chaki, S., Gurfinkel, A.: Automated assume-guarantee reasoning for omega-regular systems and specifications. Innov. Syst. Softw. Eng. **7**(2), 131–139 (2011)
34. Chalupar, G., Peherstorfer, S., Poll, E., De Ruiter, J.: Automated reverse engineering using lego®. In: Proceedings of the 8th USENIX Conference on Offensive Technologies, WOOT 2014, p. 9. USENIX Association, Berkeley (2014)
35. Chen, Y.F., Hsieh, C., Lengál, O., Lii, T.J., Tsai, M.H., Wang, B.Y., Wang, F.: PAC learning-based verification and model synthesis. In: Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, pp. 714–724. ACM, New York (2016)
36. Chen, Y.-F., Wang, B.-Y.: Learning Boolean functions incrementally. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 55–70. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_10
37. Cheng, C.-H., et al.: Algorithms for synthesizing priorities in component-based systems. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 150–167. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_12
38. Cho, C.Y., Babić, D., Poosankam, P., Chen, K.Z., Wu, E.X., Song, D.: MACE: model-inference-assisted concolic exploration for protocol and vulnerability discovery. In: Proceedings of the 20th USENIX Conference on Security, SEC 2011, p. 10. USENIX Association, Berkeley (2011)
39. Choi, W.: Automated testing of graphical user interfaces: a new algorithm and challenges. In: Proceedings of the 2013 ACM Workshop on Mobile Development Lifecycle, MobileDeLi 2013, pp. 27–28. ACM, New York (2013)
40. Choi, W., Necula, G., Sen, K.: Guided GUI testing of android apps with minimal restart and approximate learning. SIGPLAN Not. **48**(10), 623–640 (2013)
41. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Trans. Softw. Eng. **4**(3), 178–187 (1978)
42. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
43. Cobleigh, J.M., Giannakopoulou, D., PĂsĂreanu, C.S.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 331–346. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36577-X_24

44. Combe, D., de la Higuera, C., Janodet, J.-C.: Zulu: an interactive learning competition. In: Yli-Jyrä, A., Kornai, A., Sakarovitch, J., Watson, B. (eds.) FSMNLP 2009. LNCS (LNAI), vol. 6062, pp. 139–146. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14684-8_15
45. Combéfis, S., Giannakopoulou, D., Pecheur, C., Feary, M.: Learning system abstractions for human operators. In: Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering, MALETS 2011, pp. 3–10. ACM, New York (2011)
46. De Ruiter, J., Poll, E.: Protocol state fuzzing of TLS implementations. In: Proceedings of the 24th USENIX Conference on Security Symposium, SEC 2015, pp. 193–206. USENIX Association, Berkeley (2015)
47. Decker, N., Habermehl, P., Leucker, M., Thoma, D.: Learning transparent data automata. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 130–149. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07734-5_8
48. Denis, F., Lemay, A., Terlutte, A.: Residual finite state automata. Fundam. Informaticae **51**(4), 339–368 (2002)
49. Dinca, I., Ipate, F., Mierla, L., Stefanescu, A.: Learn and test for event-B – A rodin plugin. In: Derrick, J., et al. (eds.) ABZ 2012. LNCS, vol. 7316, pp. 361–364. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30885-7_32
50. Dinca, I., Ipate, F., Stefanescu, A.: Model learning and test generation for event-B decomposition. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012. LNCS, vol. 7609, pp. 539–553. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34026-0_40
51. Drews, S., D'Antoni, L.: Learning symbolic automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 173–189. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_10
52. Esparza, J., Leucker, M., Schlund, M.: Learning workflow petri nets. Fundam. Informaticae **113**(3–4), 205–228 (2011)
53. Feng, L., Han, T., Kwiatkowska, M., Parker, D.: Learning-based compositional verification for synchronous probabilistic systems. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 511–521. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_40
54. Feng, L., Kwiatkowska, M., Parker, D.: Automated learning of probabilistic assumptions for compositional reasoning. In: Giannakopoulou, D., Orejas, F. (eds.) FASE 2011. LNCS, vol. 6603, pp. 2–17. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19811-3_2
55. Fiterău-Broştean, P., Howar, F.: Learning-based testing the sliding window behavior of TCP implementations. In: Petrucci, L., Seceleanu, C., Cavalcanti, A. (eds.) FMICS/AVoCS -2017. LNCS, vol. 10471, pp. 185–200. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67113-0_12
56. Fiterău-Broştean, P., Janssen, R., Vaandrager, F.: Combining model learning and model checking to analyze TCP implementations. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 454–471. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_25
57. Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Trans. Softw. Eng. **17**(6), 591–603 (1991)
58. García, P., López, D., De Parga, M.V.: Polynomial characteristic sets for DFA identification. Theor. Comput. Sci. **448**, 41–46 (2012)

59. Garg, P., Löding, C., Madhusudan, P., Neider, D.: Learning universally quantified invariants of linear data structures. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 813–829. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_57
60. Garg, P., Löding, C., Madhusudan, P., Neider, D.: ICE: a robust framework for learning invariants. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 69–87. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_5
61. Garg, P., Neider, D., Madhusudan, P., Roth, D.: Learning invariants using decision trees and implication counterexamples. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, pp. 499–512. ACM, New York (2016)
62. Giannakopoulou, D., Rakamarić, Z., Raman, V.: Symbolic learning of component interfaces. In: Miné, A., Schmidt, D. (eds.) SAS 2012. LNCS, vol. 7460, pp. 248–264. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33125-1_18
63. Gold, E.M.: Language identification in the limit. Inf. Control **10**(5), 447–474 (1967)
64. Groz, R., Irfan, M.-N., Oriat, C.: Algorithmic improvements on regular inference of software models and perspectives for security testing. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012. LNCS, vol. 7609, pp. 444–457. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34026-0_33
65. Groz, R., Simao, A., Petrenko, A., Oriat, C.: Inferring FSM models of systems without reset. In: Bennaceur, A., Hahnle, R., Meinke, K. (eds.) ML for Dynamic Software Analysis. LNCS, vol. 11026, pp. 178–201. Springer, Cham (2018)
66. Groz, R., Simao, A., Petrenko, A., Oriat, C.: Inferring finite state machines without reset using state identification sequences. In: El-Fakih, K., Barlas, G., Yevtushenko, N. (eds.) ICTSS 2015. LNCS, vol. 9447, pp. 161–177. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25945-1_10
67. Gu, C., Roychowdhury, J.: FSM model abstraction for analog/mixed-signal circuits by learning from I/O trajectories. In: Proceedings of the 16th Asia and South Pacific Design Automation Conference, ASPDAC 2011, pp. 7–12. IEEE Press, Piscataway (2011)
68. Gulwani, S.: Automating string processing in spreadsheets using input-output examples. In: Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, pp. 317–330. ACM, New York (2011)
69. Hagerer, A., Margaria, T., Niese, O., Steffen, B., Brune, G., Ide, H.D.: Efficient regression testing of CTI-systems: testing a complex call-center solution. Ann. Rev. Commun. **55**, 1033–1040 (2001)
70. Hähnle, R., Steffen, B.: Constraint-based behavioral consistency of evolving software systems. In: Bennaceur, A., Hahnle, R., Meinke, K. (eds.) ML for Dynamic Software Analysis. LNCS, vol. 11026, pp. 205–218. Springer, Cham (2018)
71. Hungar, H., Steffen, B.: Behavior-based model construction. Int. J. Softw. Tools Technol. Transf. **6**(1), 4–14 (2004)
72. Hungar, H., Niese, O., Steffen, B.: Domain-specific optimization in automata learning. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 315–327. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_31
73. He, F., Gao, X., Wang, M., Wang, B.Y., Zhang, L.: Learning weighted assumptions for compositional verification of Markov decision processes. ACM Trans. Softw. Eng. Methodol. **25**(3), 21:1–21:39 (2016)

74. He, F., Wang, B.Y., Yin, L., Zhu, L.: Symbolic assume-guarantee reasoning through BDD learning. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, pp. 1071–1082. ACM, New York (2014)
75. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969)
76. Högberg, J.: A randomised inference algorithm for regular tree languages. Nat. Lang. Eng. **17**(2), 203–219 (2011)
77. Howar, F., Bauer, O., Merten, M., Steffen, B., Margaria, T.: The teachers' crowd: the impact of distributed oracles on active automata learning. In: Hähnle, R., Knoop, J., Margaria, T., Schreiner, D., Steffen, B. (eds.) ISoLA 2011. CCIS, pp. 232–247. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34781-8_18
78. Howar, F., Giannakopoulou, D., Rakamarić, Z.: Hybrid learning: interface generation through static, dynamic, and symbolic analysis. In: Proceedings of the 2013 International Symposium on Software Testing and Analysis, ISSTA 2013, pp. 268–279. ACM, New York (2013)
79. Howar, F., Isberner, M., Steffen, B., Bauer, O., Jonsson, B.: Inferring semantic interfaces of data structures. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012. LNCS, vol. 7609, pp. 554–571. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34026-0_41
80. Howar, F., Margaria, T., Wagner, C.: Simplifying translation validation via model extrapolation. J. Integr. Des. Process Sci. **17**(3), 71–91 (2013)
81. Howar, F., Merten, M., Steffen, B., Margaria, T.: Practical aspects of active automata learning, pp. 235–267. John Wiley and Sons, Inc. (2012)
82. Howar, F., Steffen, B., Jonsson, B., Cassel, S.: Inferring canonical register automata. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 251–266. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27940-9_17
83. Howar, F., Steffen, B., Merten, M.: From ZULU to RERS. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010. LNCS, vol. 6415, pp. 687–704. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16558-0_55
84. Howar, F., Steffen, B., Merten, M.: Automata learning with automated alphabet abstraction refinement. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 263–277. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18275-4_19
85. Hrnčič, D., Mernik, M., Bryant, B.R., Javed, F.: A memetic grammar inference algorithm for language learning. Appl. Soft Comput. **12**(3), 1006–1020 (2012)
86. Hungar, H., Margaria, T., Steffen, B.: Test-based model generation for legacy systems. In: 2003 Proceedings of the International Test Conference, ITC 2003, vol. 1, pp. 971–980, 30 September–2 October 2003
87. Ipate, F.: Learning finite cover automata from queries. J. Comput. Syst. Sci. **78**(1), 221–244 (2012)
88. Isberner, M.: Foundations of active automata learning: an algorithmic perspective. Ph.D. thesis (2015)
89. Isberner, M., Howar, F., Steffen, B.: Inferring automata with state-local alphabet abstractions. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 124–138. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_9
90. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: a redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV

2014. LNCS, vol. 8734, pp. 307–322. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_26
91. Isberner, M., Howar, F., Steffen, B.: The open-source LearnLib. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 487–495. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_32
92. Jain, S., Luo, Q., Stephan, F.: Learnability of automatic classes. J. Comput. Syst. Sci. **78**(6), 1910–1927 (2012)
93. Jain, S., Martin, E., Stephan, F.: Robust learning of automatic classes of languages. In: Kivinen, J., Szepesvári, C., Ukkonen, E., Zeugmann, T. (eds.) ALT 2011. LNCS (LNAI), vol. 6925, pp. 55–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24412-4_8
94. Jung, Y., Lee, W., Wang, B.-Y., Yi, K.: Predicate generation for learning-based quantifier-free loop invariant inference. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 205–219. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_17
95. Kasprzik, A.: Inference of residual finite-state tree automata from membership queries and finite positive data. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 476–477. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22321-1_45
96. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
97. Khalili, A., Natale, L., Tacchella, A.: Reverse engineering of middleware for verification of robot control architectures. In: Brugali, D., Broenink, J.F., Kroeger, T., MacDonald, B.A. (eds.) SIMPAR 2014. LNCS (LNAI), vol. 8810, pp. 315–326. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11900-7_27
98. Khalili, A., Tacchella, A.: Learning nondeterministic mealy machines. In: Proceedings of the 12th International Conference on Grammatical Inference, ICGI 2014, Kyoto, Japan, 17–19 September 2014, pp. 109–123 (2014)
99. Komuravelli, A., Pasareanu, C.S., Clarke, E.M.: Learning probabilistic systems from tree samples. In: Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science, LICS 2012, pp. 441–450. IEEE Computer Society, Washington, DC (2012)
100. Konev, B., Lutz, C., Ozaki, A., Wolter, F.: Exact learning of lightweight description logic ontologies. In: Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning, KR 2014, pp. 298–307. AAAI Press (2014)
101. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 23–37. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_3
102. Leucker, M., Neider, D.: Learning minimal deterministic automata from inexperienced teachers. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012. LNCS, vol. 7609, pp. 524–538. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34026-0_39
103. Lin, S.-W., André, É., Dong, J.S., Sun, J., Liu, Y.: An efficient algorithm for learning event-recording automata. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 463–472. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_35
104. Lin, S.-W., Hsiung, P.-A.: Compositional synthesis of concurrent systems through causal model checking and learning. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.)

FM 2014. LNCS, vol. 8442, pp. 416–431. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06410-9_29
105. Maler, O., Mens, I.-E.: A generic algorithm for learning symbolic automata from membership queries. In: Aceto, L., Bacci, G., Bacci, G., Ingólfsdóttir, A., Legay, A., Mardare, R. (eds.) Models, Algorithms, Logics and Tools. LNCS, vol. 10460, pp. 146–169. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63121-9_8
106. 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)
107. Margaria, T., Niese, O., Raffelt, H., Steffen, B.: Efficient test-based model generation for legacy reactive systems. In: 2004 Ninth IEEE International Proceedings of the High-Level Design Validation and Test Workshop, HLDVT 2004, pp. 95–100. IEEE Computer Society, Washington, DC (2004)
108. Medhat, R., Ramesh, S., Bonakdarpour, B., Fischmeister, S.: A framework for mining hybrid automata from input/output traces. In: Proceedings of the 12th International Conference on Embedded Software, EMSOFT 2015, pp. 177–186. IEEE Press, Piscataway (2015)
109. Meinke, K.: Learning-based testing: recent progress and future prospects. In: Bennaceur, A., Hahnle, R., Meinke, K. (eds.) ML for Dynamic Software Analysis. LNCS, vol. 11026, pp. 53–73. Springer, Cham (2018)
110. Meinke, K., Sindhu, M.A.: Incremental learning-based testing for reactive systems. In: Gogolla, M., Wolff, B. (eds.) TAP 2011. LNCS, vol. 6706, pp. 134–151. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21768-5_11
111. Meinke, K., Sindhu, M.A.: LBTest: a learning-based testing tool for reactive systems. In: Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, 18–22 March 2013, pp. 447–454 (2013)
112. Meller, Y., Grumberg, O., Yorav, K.: Learning-based compositional model checking of behavioral UML systems. In: Braga, C., Ölveczky, P.C. (eds.) FACS 2015. LNCS, vol. 9539, pp. 275–293. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-28934-2_15
113. Mens, I., Maler, O.: Learning regular languages over large ordered alphabets. Log. Methods Comput. Sci. **11**(3), 1–22 (2015)
114. Merten, M., Howar, F., Steffen, B., Cassel, S., Jonsson, B.: Demonstrating learning of register automata. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 466–471. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_32
115. Merten, M., Howar, F., Steffen, B., Pellicione, P., Tivoli, M.: Automated inference of models for black box systems based on interface descriptions. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012. LNCS, vol. 7609, pp. 79–96. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34026-0_7
116. Merten, M., Steffen, B., Howar, F., Margaria, T.: Next generation LearnLib. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 220–223. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_18
117. Shahbaz, M., Li, K., Groz, R.: Learning and integration of parameterized components through testing. In: Petrenko, A., Veanes, M., Tretmans, J., Grieskamp, W. (eds.) FATES/TestCom -2007. LNCS, vol. 4581, pp. 319–334. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73066-8_22
118. Neider, D.: Small strategies for safety games. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 306–320. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_22

119. Neider, D., Topcu, U.: An automaton learning approach to solving safety games over infinite graphs. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 204–221. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_12
120. Nerode, A.: Linear automaton transformations. Proc. Am. Math. Soc. **9**(4), 541–544 (1958)
121. Neubauer, J., Steffen, B., Bauer, O., Windmüller, S., Merten, M., Margaria, T., Howar, F.: Automated continuous quality assurance. In: Proceedings of the First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, FormSERA 2012, pp. 37–43. IEEE Press, Piscataway (2012)
122. Neubauer, J., Windmüller, S., Steffen, B.: Risk-based testing via active continuous quality control. Int. J. Softw. Tools Technol. Transf. **16**(5), 569–591 (2014)
123. Maler, O., Pnueli, A.: On the learnability of infinitary regular sets. Inf. Comput. **118**(2), 316–326 (1995)
124. Grinchtein, O., Jonsson, B., Pettersson, P.: Inference of event-recording automata using timed decision trees. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 435–449. Springer, Heidelberg (2006). https://doi.org/10.1007/11817949_29
125. Pasareanu, C.S., Giannakopoulou, D., Bobaru, M.G., Cobleigh, J.M., Barringer, H.: Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. Formal Methods Syst. Des. **32**(3), 175–205 (2008)
126. Popper, K.: The Logic of Scientific Discovery. Classics Series. Routledge, Abingdon (2002)
127. Raffelt, H., Margaria, T., Steffen, B., Merten, M.: Hybrid test of web applications with webtest. In: Proceedings of the 2008 Workshop on Testing, Analysis, and Verification of Web Services and Applications, TAV-WEB 2008, pp. 1–7. ACM, New York (2008)
128. Raffelt, H., Merten, M., Steffen, B., Margaria, T.: Dynamic testing via automata learning. Int. J. Softw. Tools Technol. Transf. **11**(4), 307–324 (2009)
129. Raffelt, H., Steffen, B., Berg, T., Margaria, T.: LearnLib: a framework for extrapolating behavioral models. Int. J. Softw. Tools Technol. Transf. **11**(5), 393–407 (2009)
130. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Inf. Comput. **103**(2), 299–347 (1993)
131. Schuts, M., Hooman, J., Vaandrager, F.: Refactoring of legacy software using model learning and equivalence checking: an industrial experience report. In: Ábrahám, E., Huisman, M. (eds.) IFM 2016. LNCS, vol. 9681, pp. 311–325. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33693-0_20
132. Shahbaz, M., Groz, R.: Inferring mealy machines. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 207–222. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05089-3_14
133. 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)
134. Cassel, S., Howar, F., Jonsson, B.: RALib: a LearnLib extension for inferring EFSMs. In: DIFTS 2015 at FMCAD 2015 (2015)
135. Staworko, S., Wieczorek, P.: Learning twig and path queries. In: Proceedings of the 15th International Conference on Database Theory, ICDT 2012, pp. 140–154. ACM, New York (2012)

136. Sun, J., Xiao, H., Liu, Y., Lin, S.W., Qin, S.: TLV: abstraction through testing, learning, and validation. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 698–709. ACM, New York (2015)
137. Berg, T., Jonsson, B., Raffelt, H.: Regular inference for state machines with parameters. In: Baresi, L., Heckel, R. (eds.) FASE 2006. LNCS, vol. 3922, pp. 107–121. Springer, Heidelberg (2006). https://doi.org/10.1007/11693017_10
138. Berg, T., Jonsson, B., Raffelt, H.: Regular inference for state machines using domains with equality tests. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 317–331. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78743-3_24
139. Bohlin, T., Jonsson, B.: Regular inference for communication protocol entities. Technical report. Department of Information Technology, Uppsala University, Schweden (2009)
140. Margaria, T., Raffelt, H., Steffen, B.: Knowledge-based relevance filtering for efficient system-level test-based model generation. Innov. Syst. Softw. Eng. **1**(2), 147–156 (2005)
141. Volpato, M., Tretmans, J.: Active learning of nondeterministic systems from an ioco perspective. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014. LNCS, vol. 8802, pp. 220–235. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45234-9_16
142. Walkinshaw, N.: Assessing test adequacy for black-box systems without specifications. In: Wolff, B., Zaïdi, F. (eds.) ICTSS 2011. LNCS, vol. 7019, pp. 209–224. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24580-0_15
143. Walkinshaw, N., Taylor, R., Derrick, J.: Inferring extended finite state machine models from software executions. Empirical Softw. Engg. **21**(3), 811–853 (2016)
144. Wang, X., Gulwani, S., Singh, R.: FIDEX: filtering spreadsheet data using examples. In: Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016, pp. 195–213. ACM, New York (2016)
145. Windmüller, S., Neubauer, J., Steffen, B., Howar, F., Bauer, O.: Active continuous quality control. In: Proceedings of the 16th International ACM Sigsoft Symposium on Component-Based Software Engineering, CBSE 2013, pp. 111–120. ACM, New York (2013)
146. Xue, Y., Wang, J., Liu, Y., Xiao, H., Sun, J., Chandramohan, M.: Detection and classification of malicious JavaScript via attack behavior modelling. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, pp. 48–59. ACM, New York (2015)