The Open-Source LearnLib

A Framework for Active Automata Learning

Malte Isberner[1], Falk Howar[2], and Bernhard Steffen[1]

[1] TU Dortmund University
D-44221 Dortmund, Germany,
{malte.isberner,steffen}@cs.tu-dortmund.de,
[2] IPSSE / TU Clausthal
D-38678 Clausthal-Zellerfeld, Germany
falk.howar@tu-clausthal.de

Abstract. In this paper, we present *LearnLib*, a library for active automata learning. The current, open-source version of *LearnLib* was completely rewritten from scratch, incorporating the lessons learned from the decade-spanning development process of the previous versions of *LearnLib*. Like its immediate predecessor, the open-source *LearnLib* is written in Java to enable a high degree of flexibility and extensibility, while at the same time providing a performance that allows for large-scale applications. Additionally, *LearnLib* provides facilities for visualizing the progress of learning algorithms in detail, thus complementing its applicability in research and industrial contexts with an educational aspect.

1 Introduction

Active automata learning, from its early beginnings almost thirty years ago [6], inspired a number of applications in quite a number of fields (see [19] for a survey). However, it took almost a decade for the software verification and testing community to recognize its value of being able to provide models of black-box systems for the plethora of model-based tools and techniques. More precisely, it was not until the seminal works of Peled *et al.* [36], employing automata learning to model check black-box systems, and Steffen *et al.* [18], who used it to automatically generate test cases for legacy computer-telephony integrated systems, that this use case of automata learning was discovered.

Since then, however, active automata learning has enjoyed quite a success story, having been used as a valuable tool in areas as diverse as automated GUI testing [13], fighting bot-nets [12], or typestate analysis [5, 41]. Most of these works, however, used their custom, one-off implementation of the well-known L^* learning algorithm [6], and hence invested relatively little effort for optimizations, or using a more sophisticated (but harder to implement and lesser-known) algorithm altogether.[3]

[3] An elaborate discussion on the theoretical aspects of active automata learning, as well as on the challenges that arise in practice, are outside the scope of this paper. We refer the interested reader to [39] for an introduction focusing on these matters.

We started developing the *LearnLib*[4] library to provide researchers and practitioners with a reusable set of components to facilitate and promote the use of active automata learning, and to enable access to cutting-edge automata learning technology. From the beginnings of the development of *LearnLib*, started in 2003, until now, more than a decade has passed. In these years, many lessons were learned on what makes for a usable, efficient and practically feasible product that fulfills this goal (cf. [25, 35, 37]).

These lessons form the basis of the new *LearnLib* presented in this paper. The new *LearnLib* is not just an overhaul of the prior version, but completely re-written from scratch. It provides a higher level of abstraction and increased flexibility, while simultaneously being the fastest version of *LearnLib* to date (cf. Section 4). As a service to the community and to encourage contributions by and collaborations with other research groups, we decided to make *LearnLib* available under an open-source license (the *Apache License, version 2.0*[5]). In the remainder of this paper we highlight two aspects that we address with *LearnLib*.

Advanced Features. This is what we consider the strongest case for preferring a comprehensive automata learning framework such as *LearnLib* over a custom implementation. While implementing the original version of L^* is not a challenging task, the situation is different for more refined active learning algorithms, such as Rivest&Schapire's [38], Kearns&Vazirani's [30] or even the very recent TTT algorithm [28]. While we found these algorithms to consistently outperform L^*, the latter remains the most widely used. Also, several other advanced optimizations such as query parallelization or efficient query caches are typically neglected. Through *LearnLib*'s modular design, changing filters, algorithm parameters or even the whole algorithm is a matter of a few lines of code, yielding valuable insights on how different algorithms perform on certain input data. Many of these features rely on *AutomataLib*, the standalone finite-state machine library that was developed for *LearnLib*, which provides a rich toolbox of data structures and algorithms for finite-state machines. The design of *AutomataLib* is presented in Section 2, while Section 3 provides a more comprehensive overview of *LearnLib*'s feature set.

Performance. The implementation of a learning algorithm comes with many performance pitfalls. Even though in most cases the time taken by the actual learning algorithm is an uncritical aspect (compared to the time spent in executing queries, which may involve, e.g., network communication), it should be kept as low as reasonably possible. Besides, an efficient management of data structures is necessary to enable learning of large-scale systems without running into out-of-memory conditions or experiencing huge performance slumps. In *LearnLib*, considerable effort was spent on efficient implementations while providing a conveniently high level of abstraction. This will be detailed in Section 4.

Finally, we conclude the paper by briefly discussing envisioned future work in Section 5.

[4] http://www.learnlib.de
[5] https://www.apache.org/licenses/LICENSE-2.0

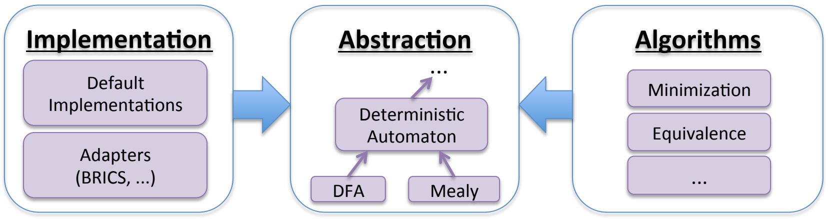

Fig. 1: Architecture of *AutomataLib*

2 AutomataLib

One of the main architectural changes of the open-source *LearnLib* is that it uses a dedicated, stand-alone library for representing and manipulating automata, called *AutomataLib*.[6] While *AutomataLib* is formally independent of *LearnLib*, its development process is closely intertwined with the one of *LearnLib*. For this reason, *AutomataLib* mainly focuses on deterministic automata, even though selected classes of non-deterministic automata are supported as well (e.g., NFAs).

AutomataLib is divided into an *abstraction layer*, automata *implementations*, and *algorithms* (cf. Fig. 1). The abstraction layer comprises a set of *Java* interfaces to represent various types of automata and graphs, organized in a complex, fine-grained type hierarchy. Furthermore, these interfaces were designed in a generic fashion, to integrate existing, third-party automata implementations into *AutomataLib*'s interface landscape with as little effort and run-time overhead as possible. For instance, a proof-of-concept adapter for the BRICS automaton library[7] could be realized in as little as 20 lines of *Java* code.

Adapters like for the BRICS library form one part of the implementation layer. The other part are generic automaton implementations, e.g., for DFAs or Mealy machines, that provide good defaults for general setups, and are also used by most algorithms in *LearnLib* to store hypotheses.

Sample algorithms shipped with *AutomataLib* include minimization, equivalence testing, or visualization (via *GraphVIZ*'s[8] `dot` tool). The set of functionalities will be continuously extended, with a strong focus on functionality either directly required in *LearnLib*, or desirable in a typical automata learning application context.

An important aspect is that the algorithms operate solely on the abstraction layer, meaning that they are implementation agnostic: they can be used with a (wrapped) BRICS automaton as well as with other automaton implementations. Furthermore, the generic design enables a high degree of code reuse: the minimization (or equivalence checking) algorithm can be used for both DFA and Mealy machines, as it is designed to only require a *deterministic automaton*, instead of a concrete machine type (or even implementation).

[6] http://www.automatalib.net/
[7] http://www.brics.dk/automaton/
[8] http://www.graphviz.org/

3 LearnLib

LearnLib provides a set of components to apply automata learning in practical settings, or to develop or analyze automata learning algorithms. These can be grouped into three main classes: learning algorithms, methods for finding counterexamples (so-called *Equivalence Queries*), and infrastructure components.

Learning Algorithms. *LearnLib* features a rich set of learning algorithms, covering the majority of algorithms which have been published (and many beyond that). Care was taken to develop the algorithms in a modular and parameterizable fashion, which allows us to use a single "base" algorithm to realize several algorithms described in the literature, e.g., by merely exchanging the involved counterexample analysis strategy. Perhaps the best example for this is the $\mathbf{L}^*$ algorithm [6], which can be configured to pose as **Maler&Pnueli's** [31], **Rivest&Schapire's** [38], or **Shahbaz's** [26] algorithm, **Suffix1by1** [26], or variants thereof. Other base algorithms available in *LearnLib* are the **Observation Pack** [21] algorithm, **Kearns&Vazirani's** [30] algorithm, the **DHC** [34] algorithm, and the **TTT** [28] algorithm. These, too, can be adapted in the way they handle counterexamples, e.g., by linear search, binary search (à la Rivest&Schapire), or exponential search [29]. With the exception of **DHC**, all these algorithms are available in both DFA and Mealy versions. Furthermore, *LearnLib* features the $\mathbf{NL}^*$ algorithm for learning NFAs [8].

Equivalence Tests and Finding Counterexamples. Once a learning algorithm converges to a stable hypothesis, a *counterexample* is needed to ensure further progress. In the context of active learning, the process of searching for a counterexample is also referred to as an *equivalence query*. "Perfect" equivalence queries are possible only when a model of the target system is available. In this case, *LearnLib* uses Hopcroft and Karp's near-linear equivalence checking algorithm [4, 20] available through *AutomataLib*. In black-box scenarios, equivalence queries can be approximated using conformance tests. *AutomataLib* provides implementations of the W-method [14] and the Wp-method [16], two of the few conformance tests that can find missing states. Often, the cheapest and fastest way of approximating equivalence queries is searching for counterexamples directly: *LearnLib* implements a random walk (only for Mealy machines), randomized generation of tests, and exhaustive generation of test inputs (up to a certain depth).

Infrastructure. The third class of components that come with *LearnLib* provide useful infrastructure functionality such as a logging facility, an import/export mechanism to store and load hypotheses, or utilities for gathering statistics. An important component for many practical applications are (optimizing) *filters*, which pre-process the queries posed by the learning algorithm. A universally useful example of such a filter is a *cache filter* [32], eliminating duplicate queries that most algorithms pose. Other examples include a parallelization component that distributes queries across multiple workers [22], a mechanism for reusing system states to reduce the number of resets [7], and for prefix-closed systems [32].

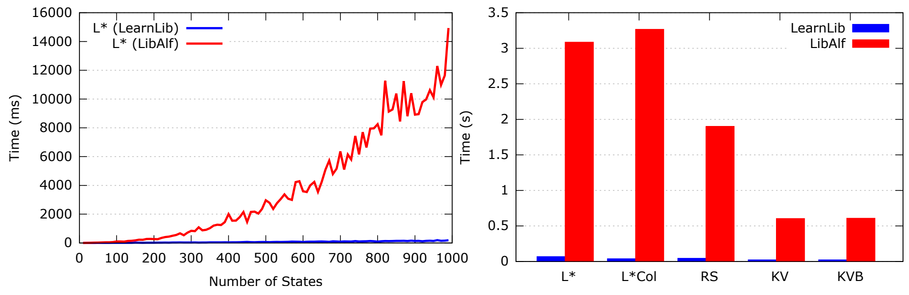

Fig. 2: Performance comparison between the new *LearnLib* and *libalf*. Left: run-time of the classic L* algorithm on a series of randomly generated automata with state counts between 10 and 1000. Right: run-time of five comparable algorithms from *LearnLib* and *libalf* on a DFA with 500 states.

For a learning algorithm to work in practice, some interface to the system under learning (SUL) needs to be available. While this is generally specific to the SUL itself, *LearnLib* provides *SUL adapters* for typical cases, e.g., *Java* classes, web-services, or processes that are interfaced with via standard I/O.

4 Evaluation

We are aware of two other open-source automata learning libraries that provide implementations of textbook algorithms, complemented by own developments:

libalf[9] The *Automata Learning Framework* [9], was developed primarily at the RWTH Aachen. It is available under LGPLv3 and written in C++. Its active development seems to have ceased; the last version was released in April 2011.

AIDE[10] The *Automata-Identification Engine*, under active development, is available under the open-source license LGPLv2.1 and written in C#.

The ambitions behind *LearnLib* go further: It is specifically designed to easily compose new custom learning algorithms on the basis of components for counterexample anaylsis, approximations of equivalence queries, as well as connectors to real life systems. Moreover, *LearnLib* provides a variety of underlying data structures, and various means for visualizing the algorithm and its statistics. This does not only facilitate the construction of highly performant custom solutions, but also provides a deeper understanding of the algorithms' characteristics. The latter has been essential, e.g., for designing the TTT algorithm [28], which almost uniformly outperforms all the previous algorithms.

Performance. As we have mentioned earlier, the open-source *LearnLib* is the fastest version of *LearnLib* to date, and moreover the fastest automata learning implementation that we are aware of. We have conducted a preliminary performance evaluation, comparing the new *LearnLib* to *libalf* and the old, closed-source version of *LearnLib* (which we will refer to as *JLearn* in order to avoid confusion). A visualization of some of the results comparing *LearnLib* and *libalf* is shown in Figure 2. It can be clearly seen that in the considered setting, *LearnLib* is more than an order of magnitude faster than *libalf* (even though the former is implemented in Java while the latter is implemented in C++). More importantly, the gap grows with the size of the system to be learned. In our experiments, the open-source *LearnLib* also outperformed *JLearn* on a similar scale. More detailed performance data can be found on the *LearnLib* website.[11]

Applications. The performance data demonstrates that *LearnLib* provides a robust basis for fast and scalable active automata learning solutions. Consequently, in its ten years of continued development, *LearnLib* has been used in a number of research and industry projects, of which we briefly present some of the more recent ones. A more complete list can be found on the *LearnLib* homepage. *LearnLib* has been used to infer models of smart card readers [11] and of bank cards [3]. The models were used to verify security properties of these systems. In [2, 15], models of communication protocols are inferred using *LearnLib*. The models are used to verify the conformance of protocol implementations to the corresponding specifications. At TU Dortmund, *LearnLib* has been used in an industry project [40] to generate models of a web application. The models were used to test regressions in the user interface and in the business processes of this application. The authors of [33] propose a method for generating checking circuits for functions implemented in FPGAs. The method uses models of the functions that are inferred with *LearnLib*. *LearnLib* is also used in other tools: PSYCO [17, 23] is a tool for generating precise interfaces of software components developed at CMU and NASA Ames. The tool combines concolic execution and active automata learning (i.e., *LearnLib*). Tomte, developed at the Radboud University of Nijmegen [1] leverages regular inference algorithms provided by *LearnLib* to infer richer classes of models by simultaneously inferring sophisticated abstractions (or "mappers").

5 Conclusion

In this paper we have presented *LearnLib*, a versatile open-source library of active automata learning algorithms. *LearnLib* is unique in its modular design, which has furthered the development of new learning algorithms (e.g., the TTT algorithm [28]) and tools (e.g., Tomte [1] and PSYCO [17, 23]).

While in many aspects the open-source *LearnLib* by far surpasses the capabilities of the previous version, there are two major features which have yet to be ported. The first is *LearnLib Studio* (cf. [35]), a graphical user interface for *LearnLib*, and the second is an extension for learning *Register Automata*. An extension for learning Register Automata with the theory of equality only was available upon request for the old *LearnLib* in binary form [24, 27]. We are currently working on a generalized approach [10], which will be included in the open-source release.

[11] http://learnlib.de/features/performance

References

1. Aarts, F., Heidarian, F., Kuppens, H., Olsen, P., Vaandrager, F.W.: Automata learning through counterexample guided abstraction refinement. In: FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings. pp. 10–27 (2012)
2. Aarts, F., Jonsson, B., Uijen, J., Vaandrager, F.W.: Generating models of infinite-state communication protocols using regular inference with abstraction. Formal Methods in System Design 46(1), 1–41 (2015)
3. Aarts, F., de Ruiter, J., Poll, E.: Formal models of bank cards for free. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Workshops Proceedings, Luxembourg, Luxembourg, March 18-22, 2013. pp. 461–468 (2013)
4. Almeida, M., Moreira, N., Reis, R.: Testing the equivalence of regular languages. In: Proceedings Eleventh International Workshop on Descriptional Complexity of Formal Systems, DCFS 2009, Magdeburg, Germany, July 6-9, 2009. pp. 47–57 (2009), http://dx.doi.org/10.4204/EPTCS.3.4
5. Alur, R., Cerný, P., Madhusudan, P., Nam, W.: Synthesis of interface specifications for java classes. In: Palsberg, J., Abadi, M. (eds.) Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005. pp. 98–109. ACM (2005), http://doi.acm.org/10.1145/1040305.1040314
6. Angluin, D.: Learning Regular Sets from Queries and Counterexamples. Inf. Comput. 75(2), 87–106 (1987)
7. Bauer, O., Neubauer, J., Steffen, B., Howar, F.: Reusing System States by Active Learning Algorithms. In: Moschitti, A., Scandariato, R. (eds.) Eternal Systems, CCSE, vol. 255, pp. 61–78. Springer-Verlag (2012)
8. Bollig, B., Habermehl, P., Kern, C., Leucker, M.: Angluin-style Learning of NFA. In: Proc. IJCAI'09. pp. 1004–1009. IJCAI'09, San Francisco, CA, USA (2009)
9. 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.) Computer Aided Verification, Lecture Notes in Computer Science, vol. 6174, pp. 360–364. Springer Berlin Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-14295-6_32
10. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Learning extended finite state machines. In: Software Engineering and Formal Methods - 12th International Conference, SEFM 2014, Grenoble, France, September 1-5, 2014. Proceedings. pp. 250–264 (2014)
11. Chalupar, G., Peherstorfer, S., Poll, E., de Ruiter, J.: Automated reverse engineering using lego®. In: 8th USENIX Workshop on Offensive Technologies, WOOT '14, San Diego, CA, USA, August 19, 2014. (2014)
12. Cho, C.Y., Babić, D., Shin, R., Song, D.: Inference and Analysis of Formal Models of Botnet Command and Control Protocols. In: Proc. CCS'10. pp. 426–440. ACM, Chicago, Illinois, USA (2010)
13. Choi, W., Necula, G., Sen, K.: Guided gui testing of android apps with minimal restart and approximate learning. In: Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages &#38; Applications. pp. 623–640. OOPSLA '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2509136.2509552

14. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Trans. Software Eng. 4(3), 178–187 (1978)
15. Fiterau-Brostean, P., Janssen, R., Vaandrager, F.W.: Learning fragments of the TCP network protocol. In: Formal Methods for Industrial Critical Systems - 19th International Conference, FMICS 2014, Florence, Italy, September 11-12, 2014. Proceedings. pp. 78–93 (2014)
16. Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. Software Engineering, IEEE Transactions on 17(6), 591–603 (1991)
17. Giannakopoulou, D., Rakamarić, Z., Raman, V.: Symbolic learning of component interfaces. In: Miné, A., Schmidt, D. (eds.) Proceedings of the 19th International Static Analysis Symposium (SAS 2012). Lecture Notes in Computer Science, vol. 7460, pp. 248–264. Springer (2012)
18. Hagerer, A., Hungar, H., Niese, O., Steffen, B.: Model generation by moderated regular extrapolation. In: Proceedings of the 5^{th} Int. Conf. on Fundamental Approaches to Software Engineering, FASE '02. Lecture Notes in Computer Science, vol. 2306, pp. 80–95. Springer Verlag (2002)
19. de la Higuera, C.: A bibliographical study of grammatical inference. Pattern Recogn. 38(9), 1332–1348 (Sep 2005), http://dx.doi.org/10.1016/j.patcog.2005.01.003
20. Hopcroft, J., Karp, R.: A linear algorithm for testing equivalence of finite automata. Tech. Rep. 0, Dept. of Computer Science, Cornell U (December 1971)
21. Howar, F.: Active Learning of Interface Programs. Ph.D. thesis, TU Dortmund University (2012), http://dx.doi.org/2003/29486
22. 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.) Proc. ISoLA 2012, pp. 232–247. CCIS, Springer (2012)
23. Howar, F., Giannakopoulou, D., Rakamarić, Z.: Hybrid learning: Interface generation through static, dynamic, and symbolic analysis. In: Proceedings of the International Symposium on Software Testing and Analysis (ISSTA). pp. 268–279. ACM (2013)
24. Howar, F., Steffen, B., Jonsson, B., Cassel, S.: Inferring Canonical Register Automata. In: Kuncak, V., Rybalchenko, A. (eds.) Proc. VMCAI'12. LNCS, vol. 7148, pp. 251–266. Springer (2012)
25. Hungar, H., Niese, O., Steffen, B.: Domain-Specific Optimization in Automata Learning. In: Proceedings of the 15^{th} Int. Conf. on Computer Aided Verification, CAV'03. Lecture Notes in Computer Science, vol. 2725, pp. 315–327. Springer Verlag (2003)
26. Irfan, M.N., Oriat, C., Groz, R.: Angluin Style Finite State Machine Inference with Non-optimal Counterexamples. In: 1st Int. Workshop on Model Inference In Testing (2010)
27. Isberner, M., Howar, F., Steffen, B.: Learning Register Automata: From Languages to Program Structures. Machine Learning 96(1-2), 65–98 (2014), http://dx.doi.org/10.1007/s10994-013-5419-7
28. Isberner, M., Howar, F., Steffen, B.: The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. In: Bonakdarpour, B., Smolka, S. (eds.) Runtime Verification, Lecture Notes in Computer Science, vol. 8734, pp. 307–322. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-11164-3_26

29. Isberner, M., Steffen, B.: An Abstract Framework for Counterexample Analysis in Active Automata Learning. In: Clark, A., Kanazawa, M., Yoshinaka, R. (eds.) Proceedings of the 12th International Conference on Grammatical Inference, ICGI 2014, Kyoto, Japan, September 17-19, 2014. JMLR Proceedings, vol. 34, pp. 79–93. JMLR.org (2014), http://jmlr.org/proceedings/papers/v34/isberner14a.html
30. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge, MA, USA (1994)
31. Maler, O., Pnueli, A.: On the Learnability of Infinitary Regular Sets. Information and Computation 118(2), 316–326 (1995)
32. Margaria, T., Raffelt, H., Steffen, B.: Knowledge-based Relevance Filtering for Efficient System-level Test-based Model Generation. Innovations in Systems and Software Engineering 1(2), 147–156 (July 2005)
33. Matuova, L., Kastil, J., Kotásek, Z.: Automatic construction of on-line checking circuits based on finite automata. In: 17th Euromicro Conference on Digital System Design, DSD 2014, Verona, Italy, August 27-29, 2014. pp. 326–332 (2014)
34. Merten, M., Howar, F., Steffen, B., Margaria, T.: Automata Learning with On-the-Fly Direct Hypothesis Construction. In: Hähnle, R., Knoop, J., Margaria, T., Schreiner, D., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification, and Validation, pp. 248–260. Communications in Computer and Information Science, Springer Berlin Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-34781-8_19
35. Merten, M., Steffen, B., Howar, F., Margaria, T.: Next Generation LearnLib. In: Proceedings of the 17^{th} Int. Conf. on Tools and algorithms for the construction and analysis of systems, TACAS'11. Lecture Notes in Computer Science, vol. 6605, pp. 220–223. Springer Verlag (2011)
36. Peled, D., Vardi, M.Y., Yannakakis, M.: Black Box Checking. In: Wu, J., Chanson, S.T., Gao, Q. (eds.) Proc. FORTE '99. pp. 225–240. Kluwer Academic (1999)
37. 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)
38. Rivest, R.L., Schapire, R.E.: Inference of Finite Futomata Using Homing Sequences. Inf. Comput. 103(2), 299–347 (1993)
39. Steffen, B., Howar, F., Merten, M.: Introduction to active automata learning from a practical perspective. In: Formal Methods for Eternal Networked Software Systems. pp. 256–296 (2011)
40. Windmüller, S., Neubauer, J., Steffen, B., Howar, F., Bauer, O.: Active continuous quality control. In: CBSE. pp. 111–120 (2013)
41. Xiao, H., Sun, J., Liu, Y., Lin, S., Sun, C.: Tzuyu: Learning stateful typestates. In: Denney, E., Bultan, T., Zeller, A. (eds.) 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013. pp. 432–442. IEEE (2013), http://dx.doi.org/10.1109/ASE.2013.6693101