ROLL 1.0: ω-Regular Language Learning Library

Yong Li[1,2][0000−0002−7301−9234], Xuechao Sun[1,2], Andrea Turrini[1,3][0000−0003−4343−9323], Yu-Fang Chen[4][0000−0003−2872−0336] and Junnan Xu[1,2]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing
[2] University of Chinese Academy of Sciences, Beijing
[3] Institute of Intelligent Software, Guangzhou
[4] Institute of Information Science, Academia Sinica, Taipei

Abstract. We present ROLL 1.0, an ω-regular language learning library with command line tools to learn and complement Büchi automata. This open source Java library implements all existing learning algorithms for the complete class of ω-regular languages. It also provides a learning-based Büchi automata complementation procedure that can be used as a baseline for automata complementation research. The tool supports both the Hanoi Omega Automata format and the BA format used by the tool RABIT. Moreover, it features an interactive Jupyter notebook environment that can be used for educational purpose.

1 Introduction

In her seminal work [3], Angluin introduced the well-known algorithm L^* to learn regular languages by means of deterministic finite automata (DFAs). In the learning setting presented in [3], there is a *teacher*, who knows the target language L, and a *learner*, whose task is to learn the target language, represented by an automaton. The learner interacts with the teacher by means of two kinds of queries: *membership queries* and *equivalence queries*. A membership query $\mathrm{MQ}(w)$ asks whether a string w belongs to L while an equivalence query $\mathrm{EQ}(A)$ asks whether the conjectured DFA A recognizes L. The teacher replies with a witness if the conjecture is incorrect otherwise the learner completes its job. This learning setting now is widely known as *active automata learning*. In recent years, active automata learning algorithms have attracted increasing attention in the computer aided verification community: it has been applied in *black-box* model checking [24], *compositional verification* [12], *program verification* [10], *error localization* [8], and *model learning* [26].

Due to the increasing importance of automata learning algorithms, many efforts have been put into the development of automata learning libraries such as libalf [6] and LearnLib [18]. However, their focus is only on automata accepting finite words, which correspond to safety properties. The ω-regular languages are the standard formalism to describe liveness properties. The problem of learning the complete class of ω-regular languages was considered open until recently, when it has been solved by Farzan *et. al* [15] and improved by Angluin *et. al* [4].

However, the research on applying ω-regular language learning algorithms for verification problems is still in its infancy. Learning algorithms for ω-regular languages are admittedly much more complicated than their finite regular language counterparts. This becomes a barrier for the researchers doing further investigations and experiments on such topics. We present ROLL 1.0, an open-source library implementing all existing learning algorithms for the complete class of ω-regular languages known in literature, which we believe can be an enabling tool for this direction of research. To the best of our knowledge, ROLL 1.0 is the only publicly available tool focusing on ω-regular language learning.

ROLL, a preliminary version of ROLL 1.0, was developed in [22] to compare the performance of different learning algorithms for Büchi automata (BAs). The main improvements made in ROLL 1.0 compared to its previous version are as follows. ROLL 1.0 rewrites the algorithms in the core part of ROLL and obtains high modularity to allow for supporting the learning algorithms for more types of ω-automata than just BAs, algorithms to be developed in future. In addition to the BA format [1,2,11], ROLL 1.0 now also supports the Hanoi Omega Automata (HOA) format [5]. Besides the learning algorithms, ROLL 1.0 also contains complementation [23] and a new language inclusion algorithm. Both of them are built on top of the BAs learning algorithms. Experiments [23] have shown that the resulting automata produced by the learning-based complementation can be much smaller than those built by structure-based algorithms [7, 9, 19, 21, 25]. Therefore, the learning-based complementation is suitable to serve as a baseline for Büchi automata complementation researches. The language inclusion checking algorithm implemented in ROLL 1.0 is based on learning and a Monte Carlo word sampling algorithm [17]. ROLL 1.0 features an interactive mode which is used in the ROLL Jupyter notebook environment. This is particularly helpful for teaching and learning how ω-regular language learning algorithms work.

2 ROLL 1.0 Architecture and Usage

ROLL 1.0 is written entirely in Java and its architecture, shown in Fig. 1, comprises two main components: the **Learning Library**, which provides all known existing learning algorithms for Büchi automata, and the **Control Center**, which uses the learning library to complete the input tasks required by the user.

Learning Library. The learning library implements all known BA learning algorithms for the full class of ω-regular languages: the $L^{\$}$ learner [15], based on DFA learning [3], and the L^{ω} learner [22], based on three canonical *family of DFAs* (FDFAs) learning algorithms [4, 22]. ROLL 1.0 supports both *observation tables* [3] and *classification trees* [20] to store membership query answers. All learning algorithms provided in ROLL 1.0 implement the `Learner` interface; their corresponding teachers implement the `Teacher` interface. Any Java object that implements `Teacher` and can decide the equivalence of two Büchi automata is a valid teacher for the BA learning algorithms. Similarly, any Java object implementing `Learner` can be used as a learner, making ROLL 1.0 easy to extend

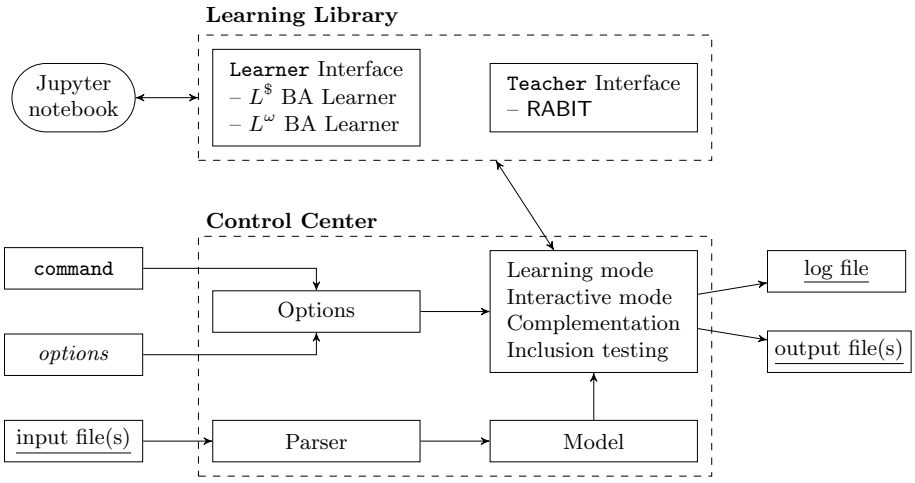

Fig. 1. Architecture of ROLL 1.0

with new learning algorithms and functionalities. The BA teacher implemented in ROLL 1.0 uses RABIT [1,2,11] to answer the equivalence queries posed by the learners since the counterexamples RABIT provides tend to be short and hence are easier to analyze; membership queries are instead answered by implementing the ASCC algorithm from [16].

Control Center. The control center is responsible for calling the appropriate learning algorithm according to the user's command and options given at command line, which is used to set the `Options`. The file formats supported by ROLL 1.0 for the input automata are the RABIT BA format [1,2,11] and the standard Hanoi Omega Automata (HOA) format [5], identified by the file extensions .ba and .hoa, respectively. Besides managing the different execution modes, which are presented below, the control center allows for saving the learned automaton into a given file (option *-out*), for further processing, and to save execution details in a log file (option *-log*). The output automaton is generated in the same format of the input. The standard way to call ROLL 1.0 from command line is

`java -jar ROLL.jar command` input file(s) [*options*]

Learning mode (command `learn`) makes ROLL 1.0 learn a Büchi automaton equivalent to the given Büchi automaton; this can be used, for instance, to get a possibly smaller BA. The default option for storing answers to membership queries is *-table*, which selects the observation tables; classification trees can be chosen instead by means of the *-tree* option.

`java -jar ROLL.jar learn` aut.hoa

for instance runs ROLL 1.0 in learning mode against the input BA aut.hoa; it learns aut.hoa by means of the L^{ω} learner using observation tables. The

three canonical FDFA learning algorithms given in [4] can be chosen by means of the options *-syntactic* (default), *-recurrent*, and *-periodic*. Options *-under* (default) and *-over* control which approximation is used in the L^ω learner [22] to transform an FDFA to a BA. By giving the option *-ldollar*, ROLL 1.0 switches to use the $L^\$$ learner instead of the default L^ω learner.

Interactive mode (command `play`) allows users to play as the teacher guiding ROLL 1.0 in learning the language they have in mind. To show how the learning procedure works, ROLL 1.0 outputs each intermediate result in the Graphviz dot layout format[1]; users can use Graphviz's tools to get a graphical view of the output BA so to decide whether it is the right conjecture.

Complementation (command `complement`) of the BA $\mathcal{B}$ in ROLL 1.0 is based on the algorithm from [23] which learns the complement automaton $\mathcal{B}^{\mathsf{c}}$ from a teacher who knows the language $\Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$. This allows ROLL 1.0 to disentangle $\mathcal{B}^{\mathsf{c}}$ from the structure of $\mathcal{B}$, avoiding the $\Omega((0.76n)^n)$ blowup [27] of the structure-based complementation algorithms (see., e.g., [7,19,21,25]).

Inclusion testing (command `include`) between two BAs $\mathcal{A}$ and $\mathcal{B}$ is implemented in ROLL 1.0 as follows: 1) first, sample several ω-words $w \in \mathcal{L}(\mathcal{A})$ and check whether $w \notin \mathcal{L}(\mathcal{B})$ to prove $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$; 2) then, try simulation techniques [11, 13, 14] to prove inclusion; 3) finally, use the learning based complementation algorithm to check inclusion. The ROLL 1.0's ω-word sampling algorithm is an extension of the one proposed in [17]. The latter only samples paths visiting any state at most twice while ROLL 1.0's variant allows for sampling paths visiting any state at most K times, where K is usually set to the number of states in $\mathcal{A}$. In this way, ROLL 1.0 can get a larger set of ω-words accepted by $\mathcal{A}$ than the set from the original algorithm.

Online availability of ROLL 1.0. ROLL 1.0 is an open-source library freely available online at https://iscasmc.ios.ac.cn/roll/, where more details are provided about its commands and options, its use as a Java library, and its GitHub repository[2]. Moreover, from the roll page, it is possible to access an online Jupyter notebook[3] allowing to interact with ROLL 1.0 without having to download and compile it. Each client gets a new instance of the notebook, provided by JupyterHub[4], so to avoid unexpected interactions between different users. Fig. 2 shows few screenshots of the notebook for learning in interactive mode the language $\Sigma^* \cdot b^\omega$ over the alphabet $\Sigma = \{a, b\}$. As we can see, the membership query $\mathrm{MQ}(w)$ is answered by means of the `mqOracle` function: it gets as input two finite words, the `stem` and the `loop` of the ultimately periodic word w, and it checks whether `loop` contains only b. Then one can create a BA learner with the oracle `mqOracle`, say the BA learner `nbaLearner`, based on observation tables and the recurrent FDFAs, as shown in the top-left screenshot. One can check the internal table structures of `nbaLearner` by printing out the learner, as in

[1] https://www.graphviz.org/
[2] https://github.com/ISCAS-PMC/roll-library
[3] https://iscasmc.ios.ac.cn/roll/jupyter
[4] https://jupyterhub.readthedocs.io/

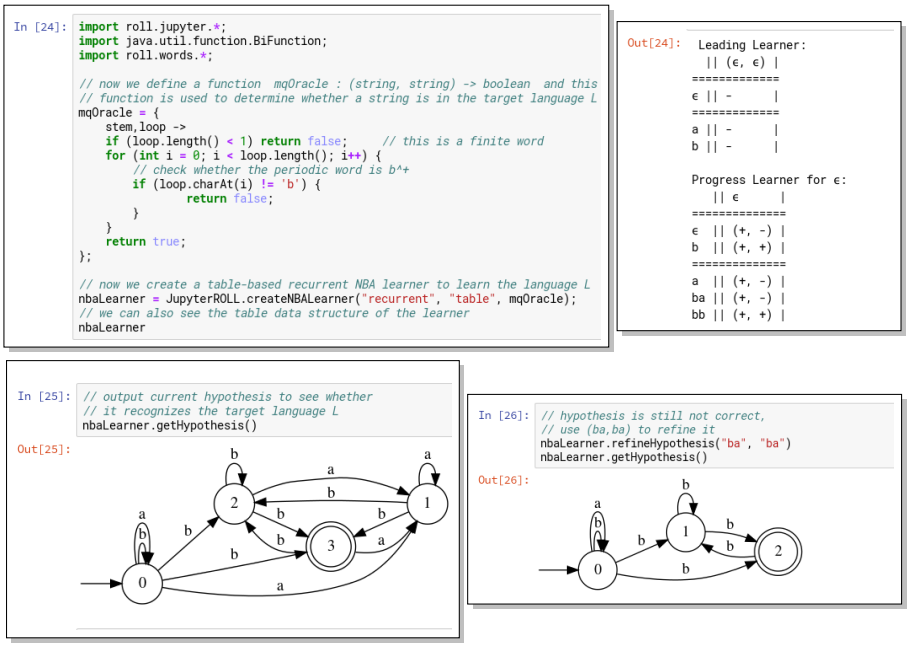

Fig. 2. ROLL 1.0 running in the Jupyter notebook for interactively learning $\Sigma^* \cdot b^\omega$

the top-right screenshot. The answer to an equivalence query is split in two parts: first, the call to `getHypothesis()` shows the currently conjectured BA; then, the call to `refineHypothesis("ba", "ba")` simulates a negative answer with counterexample $ba \cdot (ba)^\omega$. After the refinement by `nbaLearner`, the new conjectured BA is already the right conjecture.

Acknowledgement This work has been supported by the National Natural Science Foundation of China (Grants Nos. 61532019, 61761136011) and by the CAP project GZ1023.

References

1. P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in Ramsey-based Büchi automata universality and inclusion testing. In *CAV*, pages 132–147, 2010.
2. P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In *CONCUR*, pages 187–202, 2011.
3. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
4. D. Angluin and D. Fisman. Learning regular omega languages. *Theoretical Computer Science*, 650:57–72, 2016.

5. T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Křetínský, D. Müller, D. Parker, and J. Strejček. The Hanoi Omega-Automata Format. In *CAV*, pages 479–486, 2015.
6. B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, and D. R. Piegdon. libalf: The automata learning framework. In *CAV*, pages 360–364, 2010.
7. J. R. Büchi. On a decision method in restricted second order arithmetic. In *Int. Congress on Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.
8. M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, and M. Tautschnig. Learning the language of error. In *ATVA*, pages 114–130, 2015.
9. Y.-F. Chen, M. Heizmann, O. Lengál, Y. Li, M.-H. Tsai, A. Turrini, and L. Zhang. Advanced automata-based algorithms for program termination checking. In *PLDI*, pages 135–150, 2018.
10. Y.-F. Chen, C. Hsieh, O. Lengál, T.-J. Lii, M.-H. Tsai, B.-Y. Wang, and F. Wang. PAC learning-based verification and model synthesis. In *ICSE*, pages 714–724, 2016.
11. L. Clemente and R. Mayr. Advanced Automata Minimization. In *POPL*, pages 63–74, 2013.
12. J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In *TACAS*, pages 331–346, 2003.
13. D. L. Dill, A. J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation preorders. In *CAV*, pages 255–265, 1991.
14. K. Etessami, T. Wilke, and R. A. Schuller. Fair simulation relations, parity games, and state space reduction for Büchi automata. In *ICALP*, pages 694–707, 2001.
15. A. Farzan, Y.-F. Chen, E. M. Clarke, Y.-K. Tsay, and B.-Y. Wang. Extending automated compositional verification to the full class of omega-regular languages. In *TACAS*, pages 2–17, 2008.
16. A. Gaiser and S. Schwoon. Comparison of algorithms for checking emptiness of Büchi automata. In *MEMICS*, 2009.
17. R. Grosu and S. A. Smolka. Monte carlo model checking. In *TACAS*, pages 271–286, 2005.
18. M. Isberner, F. Howar, and B. Steffen. The open-source LearnLib. In *CAV*, pages 487–495, 2015.
19. D. Kähler and T. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *ICALP*, pages 724–735, 2008.
20. M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.
21. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Logic*, 2(3):408–429, 2001.
22. Y. Li, Y.-F. Chen, L. Zhang, and D. Liu. A novel learning algorithm for Büchi automata based on family of DFAs and classification trees. In *TACAS*, pages 208–226, 2017.
23. Y. Li, A. Turrini, L. Zhang, and S. Schewe. Learning to complement Büchi automata. In *VMCAI*, pages 313–335, 2018.
24. D. Peled, M. Y. Vardi, and M. Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics*, 7(2):225–246, 2001.
25. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264, 2006.
26. F. Vaandrager. Model Learning. *Communications of the ACM*, 60(2):86–95, 2017.
27. Q. Yan. Lower bounds for complementation of omega-automata via the full automata technique. *Logical Methods in Computer Science*, 4(1), 2008.